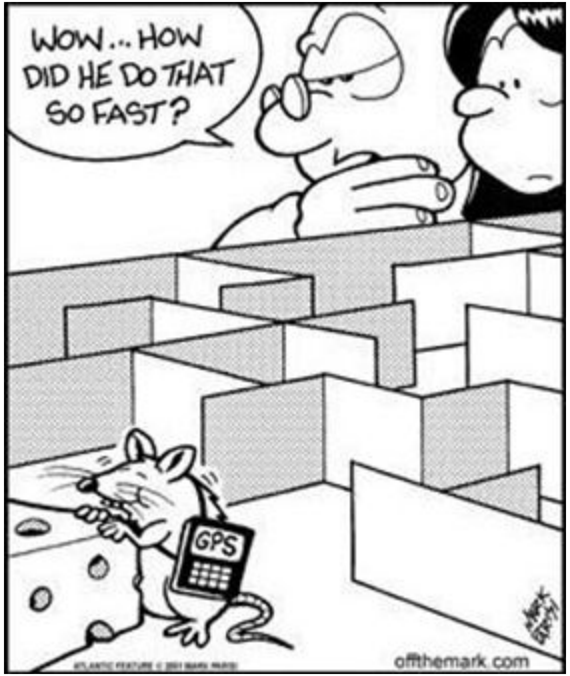
Reinforcement Learning 101

Sep 2019

Aleksandra Faust

Research Scientist

Robotics at Google



Reinforcement learning 101

- Let's talk about decision-making!
- Markov decision processes
- RL Basics
  - Q-learning
  - Components of RL
- Function approximations
  - RL in continuous spaces
  - Approximate Q-iteration
  - Deep RL
- [Wed] RL with continuous actions
  - DDPG, Soft actor critic
- [Wed] How does RL fit in ML world?
  - Inverse RL, AutoRL

Making **good** decisions.

Storage problems

- How much energy to store in a battery to handle the volatility of wind and spot prices to meet demands?

Storage problems

- How much money should we hold in cash given variable market returns and interest rates to meet the needs of a business?

Stock prices

Bonds

Elements of a storage problem

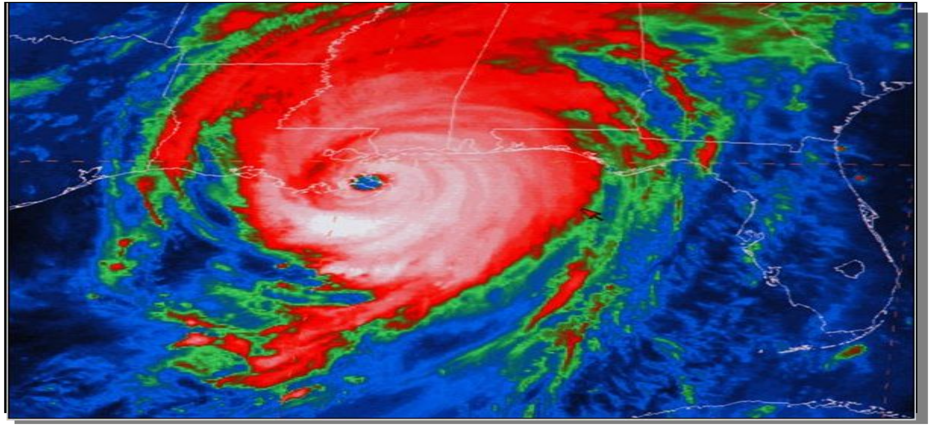
- **Controllable variable** giving the amount in storage
  - Deposit money, charge a battery, chill the water, release water from the reservoir.
  - How much energy, money, etc.?
  - **Action that the decision-maker takes.**
- Multidimensional "**state of the world**" variable that evolves exogenously
  - Prices, Interest rates, Weather, Demand/loads
  - **Action influences the state of the world.**

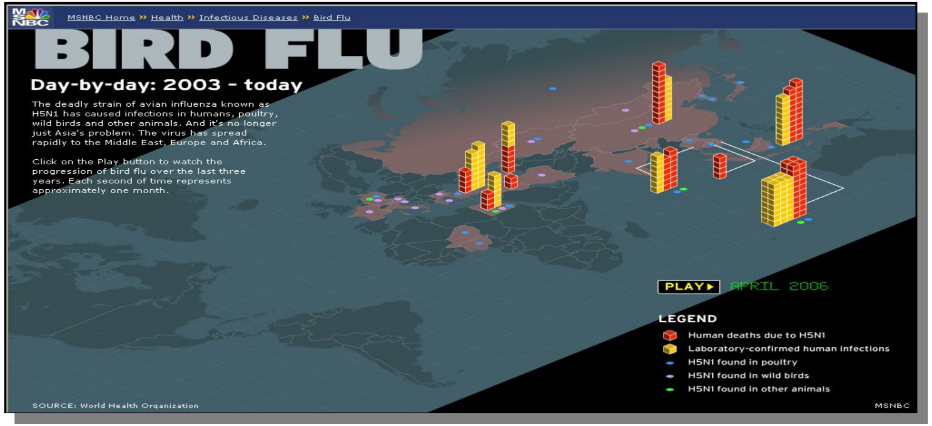
- Immediate observable feedback
  - Cost or **reward**: trading fees, cost loosing charge, cost of empty battery
- Other features
  - Problem may be time-dependent (and finite horizon) or stationary
  - We may have access to forecasts of future

Planning for a risky world

Weather

- Robust design of emergency response networks.
- Insurance prices - Design of financial instruments to hedge against weather emergencies to protect individuals, companies and municipalities.
- Design of sensor networks and communication systems to manage responses to major weather events.





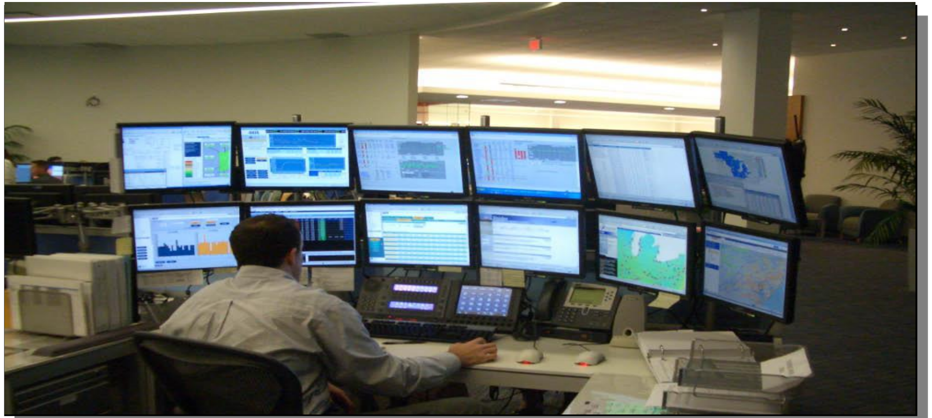
Disease

- Models of disease propagation for response planning.
- Management of medical personnel, equipment and vaccines to respond to a disease outbreak.
- Robust design of supply chains to mitigate the disruption of transportation systems.

Energy management

Energy resource allocation

- What is the right mix of energy technologies?
- How should the use of different energy resources be coordinated over space and time?
- What should my energy R&D portfolio look like?
- What is the impact of a carbon tax?



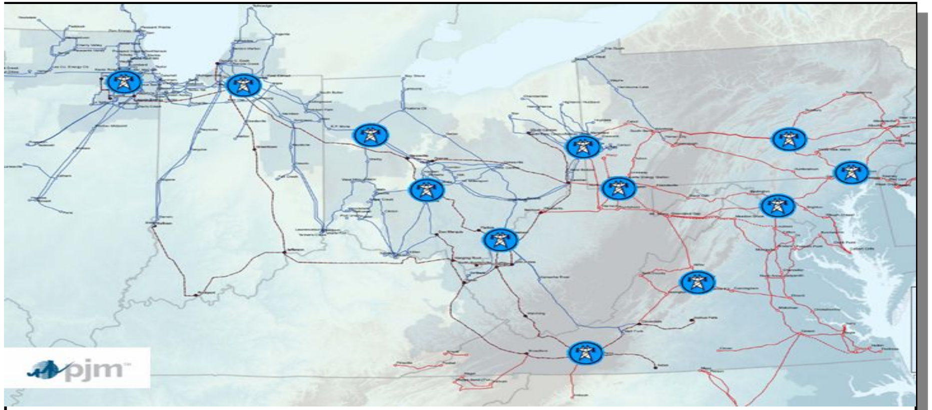
Energy markets

- How should I hedge energy commodities?
- How do I price energy assets?
- What is the right price for energy futures?

High value spare parts

Electric Power Grid

- PJM oversees an aging investment in high-voltage transformers.
- Replacement strategy needs to anticipate a bulge in retirements and failures
- 1-2 year lag times on orders. Typical cost of a replacement ~ $5 million.
- Failures vary widely in terms of economic impact on the network.



Spare parts for business jets

- ADP is used to determine purchasing and allocation strategies for over 400 high value spare parts.
- Inventory strategy has to determine what to buy, when and where to store it. Many parts are very low volume (e.g. 7 spares spread across 15 service centers).
- Inventories have to meet global targets on level of service and inventory costs.

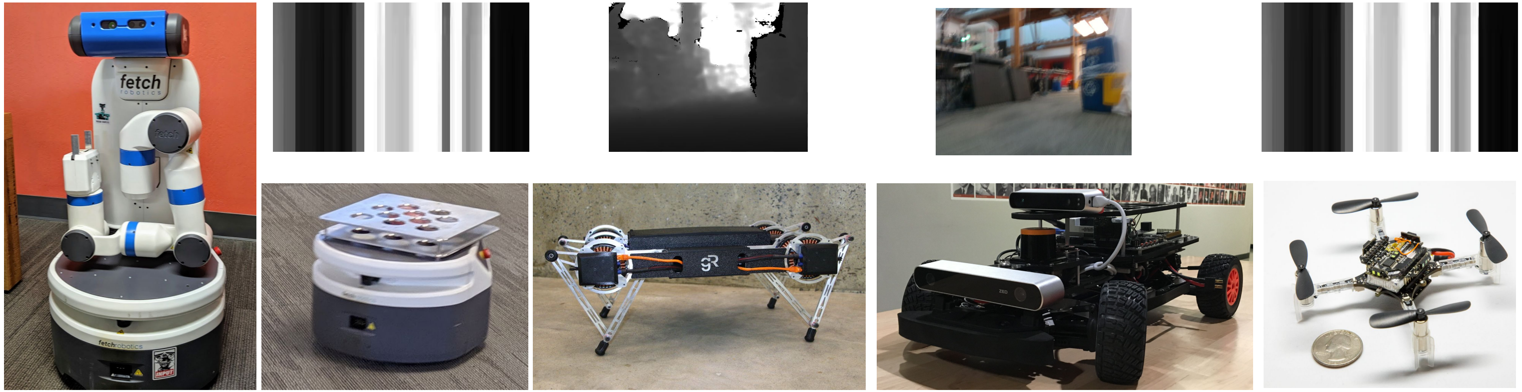
Elements of a planning problem

- **Controllable variable** specifying capacity
  - Evacuation routes, select energy portfolio…
  - **Action that the decision-maker takes.**

- Multidimensional "**state of the world**" variable that evolves exogenously
  - Prices, Weather, Demand/loads
  - **Action influences the state of the world.**

- Immediate observable feedback
  - Cost or **reward**: cost of parts' storage, number of affected people.
- Other features
  - Problem may be time-dependent (and finite horizon) or stationary
  - We may have access to forecasts of future

Systems’ control and motion planning



Fetch | Freight | Minitaur | Car Racer | CrazyFlie

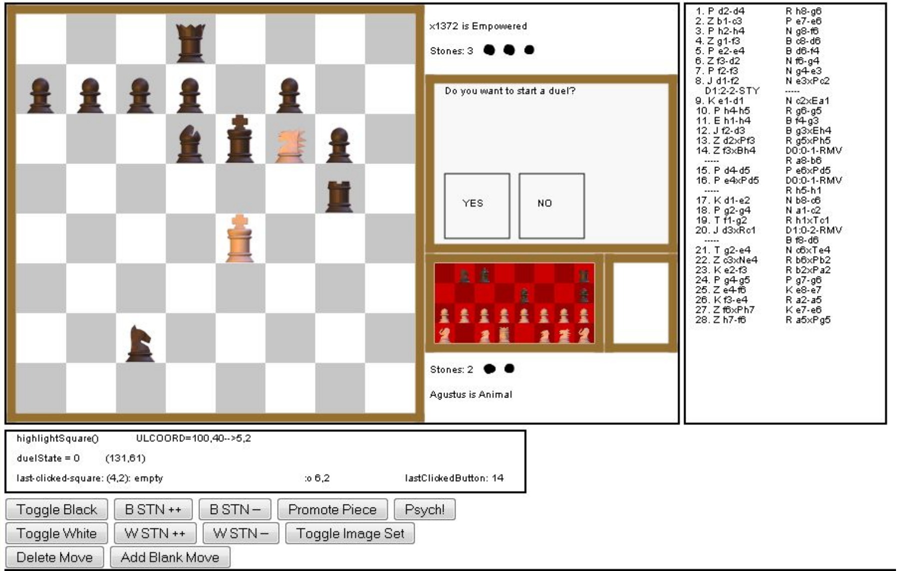
Differential drive.
Noisy sensors.
Inertia.

Kinodynamic constraints.
Noisy sensors.

< 200 KB of RAM

Artificial Intelligence

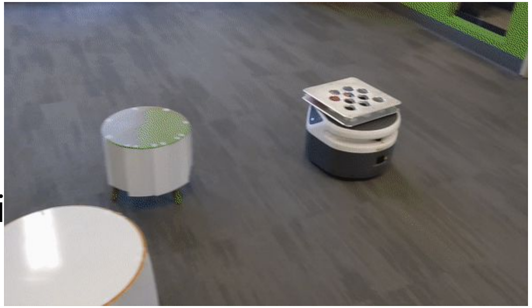
- Games
- Modelling
- Virtual reality



Elements of a motion planning problem

- **Controllable variable** moving the robot / agent
  - Move a robot or win a game pieces…
  - **Action that the decision-maker takes.**

- Multidimensional "**state of the world**" variable that evolves exogenously
  - Position, Velocity
  - **Action influences the changes in the world.**
- Immediate observable feedback
  - Cost or **reward**: game won, task completed….
- Other features:
  - Problem may be time-dependent (and finite horizon) or stati
  - We may have access to forecasts of future



Elements of a (sequential) decision-making problem

- **Controllable variable**
  - **Action that the decision-maker takes.**
- Multidimensional “**state of the world**” variable that evolves exogenously
  - **Action influences the changes in the world.**
- Immediate observable feedback
  - Cost or **reward**.
- Other features:
  - Problem may be time-dependent (and finite horizon) or stationary
  - We may have access to forecasts of future
- **Goal:**
  - **Select actions that maximize total cumulative future reward.**

Time span

- Real-time control
  - Scheduling aircraft, pilots, generators, tankers
  - Pricing stocks, options
  - Electricity resource allocation
  - Robotics
- Near-term tactical planning
  - Can I accept a customer request?
  - Should I lease equipment?
  - How much energy can I commit with my windmills?
- Strategic planning
  - What is the right equipment mix?
  - What is the value of this contract?
  - What is the value of more reliable aircraft?

Disciplines

- Sequential decisions in
  - Economics
  - Robotics
  - Energy management
  - Resource management
  - Video games/simulations
- Fields
  - Operations research
  - Systems controls
  - AI/Machine Learning
  - Neuroscience

Markov Decision Process (MDP)

Markov decision process (MDP)

- Stochastic automata with utilities
- Describes the problem
- Defined with
  - Set of states S
  - Set of actions A
  - Description of effect of each action on every state
  - Reward function R: S->*R*
- Markovian property
  - The effects of the action taken in a state, depend only on the current state, and not on how that state was reached
- Goal
  - Select actions that maximize total cumulative future reward.

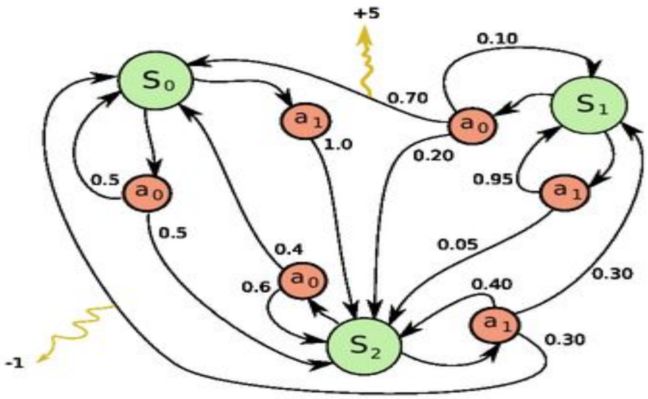
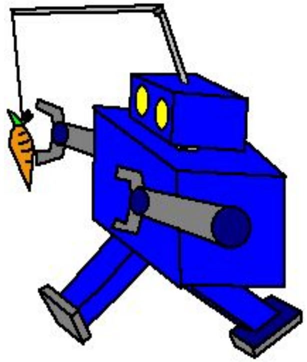


Image from Wikipedia

Learning by Reinforcement

- Questions
  - Can we influence system behavior with reinforcement?
  - Given natural consequences, can we learn to control system?
- Robot (agent)
  - Sensors observe ***state*** of environment
  - Performs ***actions*** to change the states
  - Receives numerical feedback, ***reward***
- **Problem statement**
  - **Learn to choose actions that maximize accumulated reward, i.e.**
  - **Learn the total accumulated reward to be encountered heuristic, and use hill-climbing**



MDP action description

- Deterministic
  - For each state and action the next state is unique F:SxA -> S
  - Boolean tabular representation for finite states and actions
  - Deterministic finite state automata
- Stochastic
  - For each state and action a probability distribution defines next state P(s' | s, a)
  - Numeric tabular representation for finite states and actions
  - Nondeterministic finite state automata
- Markov chain
  - Has single action (no choice)
  - Has no reward (no motivation)

MDP solution

- Policy π, mapping from states to actions
  - Tells us what action to take
  - Tabular representation for finite states and actions
- Following policy, π,
  1. Repeat
     1. Determine current state s
     2. Look up action a = π(s)
     3. Apply action a to state s
- Full observability – current state s can be fully determined, known to the system
- Partial observability – partially observable MDP (POMDP)

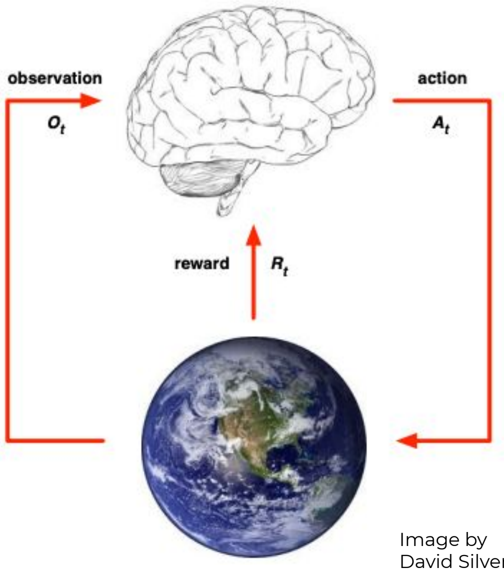


Image by David Silver

Policy evaluation

- How good is a policy?
- Deterministic
  - Sum of rewards encountered
- Stochastic
  - Expected total reward
- Objective function (value function)
  - $V(s_0) = R(s_0) + R(s_1) + R(s_2) + \ldots.$
- Can be infinite
  - Finite horizon – consider fixed number of steps
    - $V(s_0) = R(s_0) + R(s_1) + R(s_2) + \ldots. + R(s_n)$
  - Discounting – value earlier reward more
    - $V(s_0) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots. = \sum_i \gamma^i R(s_i)$

Value function (State value function)

- Goal to maximize (discounted) value function
  - $V(s_0) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ... = \sum_i \gamma^i R(s_i)$
- Optimal value function is a fixed point
  - $$\begin{aligned} V^*(s_0) &= \max V(s_0) \\ &= \max(R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots.) \\ &= R(s_0) + \gamma \max(R(s_1) + \gamma^1 R(s_2) + \ldots.) \\ &= R(s_0) + \gamma V^*(s_1) \end{aligned}$$
- Bellman equation *[Bellman, 1957]*
  - $V^*(s_0) = R(s_0) + \gamma V^*(s_1)$
  - Tool how to calculate value iteratively

Action value function (Q function)

- Q: S x A -> R
- Q(s,a) - Value of action a performed at state s.
- Optimal value function is a fixed point
  - $Q(s,a) = r + \gamma V(s')$ and $V(s) = max_a Q(s,a)$
  - $Q(s,a) = r + \gamma(max_{a'}(Q(s',a'))$
- Tool how to calculate value iteratively when (s, a, r, s') samples available.
- Optimal policy
  - π*(s) = argmax Q*(s,a), over all a
  - π*(s) = argmax V*(s'), over all state s' reachable from s

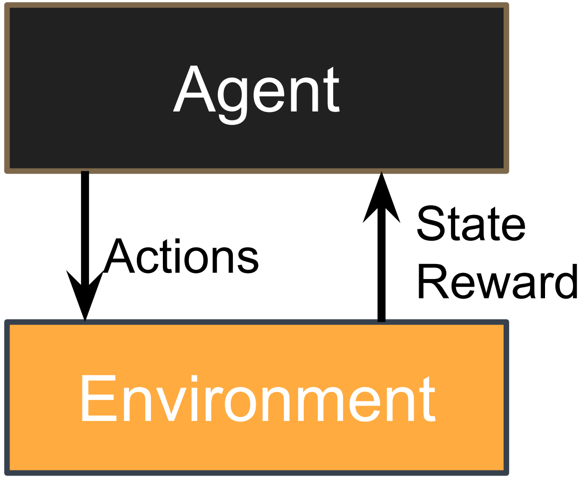
Review: Sequential Decision Making

- What are decision-making problems?
  - Problems where we modify one or more variables that influence the system to accomplish a goal.
- How do we model decision-making problems?
  - States
  - Actions
  - Transition function
  - Reward
  - Markov decision process
- What is the goal?
  - To maximize long-term gain, while making short-term decisions.
- What is a solution?
  - Policy, mapping from states to actions

Basic RL

Reinforcement Learning

- Markov decision process (MDP)
  - Set of states S
  - Set of actions A
  - Transition function F: SxA->S (**not available analytically**)
  - Reward function R: S->*R* (**not available analytically**)
- Heuristic
  - Value functions V and Q: cost to go, potential for accumulated reward
  - Example: value of high school degree
- Strategy / Goal
  - Find policy π which produces action sequence that maximizes the value
- F and R are known - dynamic programming



Example

- Find the cheese and avoid the cat
- Move up, down, left, and right

- Cheese: 100
- Cat: -1000
- Step: -1

- Mouse doesn't know:
    - Where the cheese is
    - Where the cats are

Example – dynamic programming

- Reward and transitions known

-1	-1	-1	-1	-1
-1	-1	-1000	-1	-1
-1	-1000	100	-1000	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1000	100

Reward

	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	
	-1			-1			-1000			-1			-1	
	-1			-1			-1			-1			-1	
	-1	-1	-1	-1	-1000	-1	-1000	-1	-1000	-1	-1	-1	-1	
	-1			-1000			100			-1000			-1	
	-1			-1			-1000			-1			-1	
	-1	-1000	-1	-1000	100	-1000	100	-1000	100	-1000	-1	-1000	-1	
	-1			-1			-1			-1			-1	
	-1			-1000			100			-1000			-1	
	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	
	-1			-1			-1			-1			-1	
	-1			-1			-1			-1			-1	
	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	
	-1			-1			-1			-1000			100	
	-1			-1			-1			-1			-1	
	-1	-1	-1	-1	-1	-1	-1	-1000	-1	-1000	100	-1000	100	

1^{st} iteration

Example – dynamic programming

- Reward and transitions known

	-1	-2	-2	-1	-2	-2	-1	-2	-2	-1	-2	-2	-1	
	-2			-2			-900			-2			-2	
	-2			-2			-2			-2			-2	
	-1	-2	-2	-1	-900	-2	100	-2	-900	-1	-2	-2	-1	
	-2			-900			99			-900			-2	
	-2			-2			-900			-2			-2	
	-1	-900	-2	100	99	-900	-1	-900	99	100	-2	-900	-1	
	-2			-2			99			-2			-2	
	-2			-900			99			-900			-2	
	-1	-2	-2	-1	99	-2	100	-2	99	-1	-2	-2	-1	
	-2			-2			-2			-2			99	
	-2			-2			99			-2			-2	
	-1	-2	-2	-1	-2	-2	-1	-2	-2	-1	99	-2	100	
	-2			-2			-2			-900			99	
	-2			-2			-2			-2			99	
	-1	-2	-2	-1	-2	-2	-1	-900	-2	100	99	-900	-1	

2nd iteration

	-2	-3	-3	-2	-3	-3	-2	-3	-3	-2	-3	-3	-2	
	-3			-3			-901			-3			-3	
	-3			-3			-3			-3			-3	
	-2	-3	-3	-2	-901	-3	99	-3	-901	-2	-3	-3	-2	
	-3			-901			199			-901			-3	
	-3			-3			-901			-3			-3	
	-2	-901	-3	99	199	-901	99	-901	199	99	-3	-901	-2	
	-3			98			98			98			98	
	-3			-901			199			-901			-3	
	-2	98	-3	99	98	98	99	98	98	99	98	98	99	
	-3			-3			98			98			98	
	-3			98			98			98			98	
	-2	-3	-3	-2	98	-3	99	98	98	99	98	98	99	
	-3			-3			-3			-901			199	
	-3			-3			98			98			98	
	-2	-3	-3	-2	-3	-3	-2	-901	-3	99	199	-901	99	

3rd iteration

Starts to back propagate

Example – dynamic programming

- Reward and transitions known
- Heuristic (value function) is table
- Pattern emerges
    - Cats are blocked off
    - Small area of uncertainty
- What to do
    - with no knowledge of reward?
    - when we don't know the transitions?
- Learn from experience by trial and error

-3	-4	-4	-3	-4	-4	-3	-4	-4	-3	-4	-4	-3
-4			-4			-801			-4			-4
-4			-4			-4			-4			-4
-3	-4	-4	-3	-801	-4	199	-4	-801	-3	-4	-4	-3
-4			-801			198			-801			97
-4			-4			-801			-4			-4
-3	-801	-4	199	198	-801	98	-801	198	199	97	-801	98
97			97			198			97			97
-4			-801			198			-801			97
98	97	97	98	198	97	199	97	198	98	97	97	98
-4			97			97			97			198
97			97			198			97			97
-3	97	-4	98	97	97	98	97	97	98	198	97	199
-4			-4			97			-801			198
-4			97			97			97			198
-3	-4	-4	-3	97	-4	98	-801	97	199	198	-801	98

4^{th} iteration

Example

- Cheese: 100
- Cat: -1000
- Step: -1
- Mouse doesn’t know:
  - Where the cheese is
  - Where the cats are

Action	Reward	Value
Right	-1	-1

Example

- Cheese: 100
- Cat: -1000
- Step: -1
- Mouse doesn't know:
  - Where the cheese is
  - Where the cats are

Action	Reward	Value
Right	-1	-1
Right	-1	-2

Example

- Cheese: 100
- Cat: -1000
- Step: -1
- Mouse doesn't know:
  - Where the cheese is
  - Where the cats are

Action	Reward	Value
Right	-1	-1
Right	-1	-2
Right	-1000	-1002

Example

- Cheese: 100
- Cat: -1000
- Step: -1
- Mouse doesn’t know:
  - Where the cheese is
  - Where the cats are

Action	Reward	Value
Right	-1	-1
Right	-1	-2
Right	-1000	-1002
Right	100	-902

Example

- Cheese: 100
- Cat: -1000
- Step: -1

Action	Reward	Value
Right	-1	-1
Right	-1	-2
Right	-1000	-1002
Right	100	-902
Up	-1	-903
Left	-1	-904
Up	-1	-905
Left	-1	-906

				
				
				
		-1	-1	
			-1	-1
0	-1	-1	-1000	100

Example - Continued

- How good is a particular decision?

- Estimate value of action:
  - One step at the time
  - Reward plus the value of the next state
  - What's college going to cost me plus expected earning afterwards

- State action value, Q
- Q(s,a) = r + max Q(s',a)
- max Q(s,a): value of state s

				
				
				
		r = -1	r = -1 Q(l) = -2	
			r = -1 Q(u) = -2	r = -1 Q(l)= -1
r = 0 Q(r)=-1	r=-1 Q(r)=-2	r=-1 Q(r)=-1001	r=-1000 Q(r)=-900	r= 100 Q(u)= 99

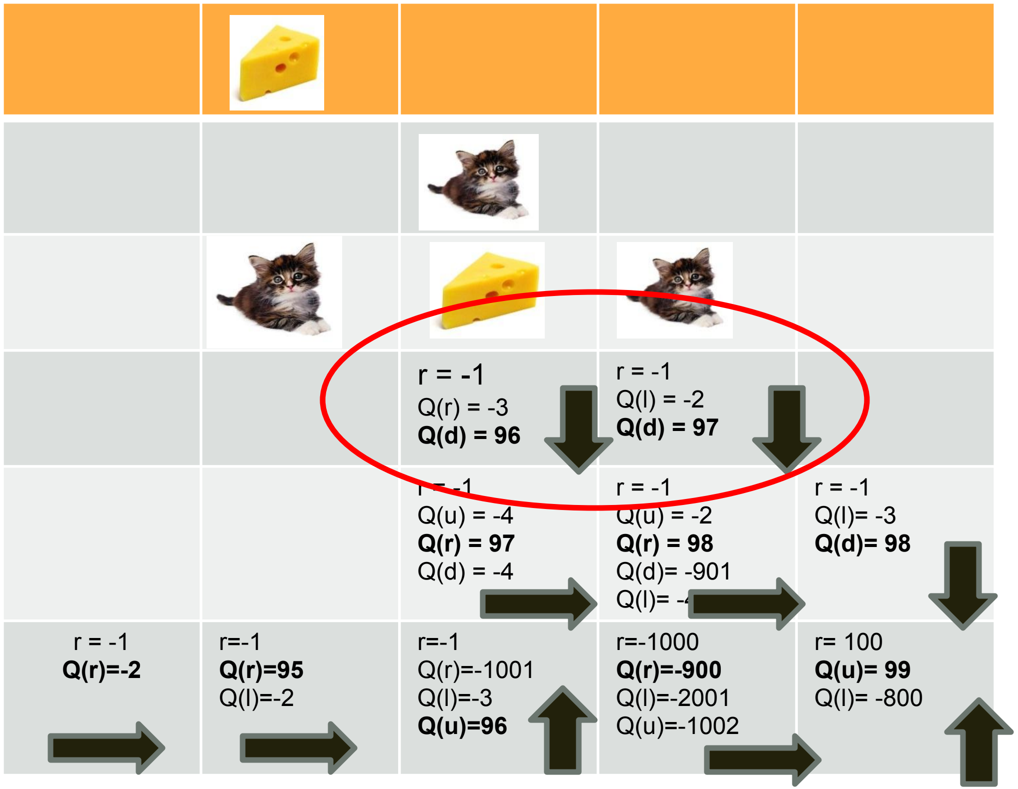
Example - Continued

- Keep exploring
- Iteratively update Q
    - Q(s,a) = r + max Q(s',a)
- The values start backtracking
- Retrospective

				
				
				
		r = -1 Q(r) = -3	r = -1 Q(l) = -2 Q(d) = -3	
			r = -1 Q(u) = -2 Q(r) = 97	r = -1 Q(l)= -2 Q(d)= 98
r = 0 Q(r)=-1	r=-1 Q(r)=-2 Q(l)=-2	r=-1 Q(r)=-1001 Q(l)=-3	r=-1000 Q(r)=-900 Q(l)=-2001	r= 100 Q(u)= 99 Q(l)= -800

Example - Continued

- After lots of exploration learns to avoid the cat
- Each state has a preferred direction: policy
- State action values might not be accurate, but direction is – most of the time
- Similar pattern, but only small area covered
- What happened?



Q-Learning

Q-Learning

- Initialize Q table
- Repeat
  - Pick a state, action (s,a) transition
  - Make the transition from (s,a) -> s'
  - Receive reward r
  - Update $Q(s,a) \leftarrow (1-\alpha)Q(s, a) + \alpha(r + \gamma \max_a Q(s', a))$
- Until sufficiently happy with performance

[Watkins 1989]

Q-Learning Continued

- Update done as weighted average – Why?
- α – learning rate ($0 <= \alpha <= 1$)
  - Weight of new information
- γ – discount factor ($0 <= \gamma < 1$)
  - Instant gratification: rewards obtained sooner are more desirable
  - Keeps value function finite
- Converges to optimal policy when
  - Repeated infinitely many times
  - Explores
- Works in discrete (small) spaces
  - But it is slow

Repeat
    Pick (s,a)
    Make the transition to s'
    Receive reward r
    $Q(s,a) \leftarrow (1-\alpha)Q(s, a) + \alpha(r + \gamma \max Q(s', a))$
Until sufficiently happy with performance

Exploration vs. Exploitation

- Exploration
  - Discovers state space
  - Takes risks, makes mistakes, learns
  - Random, biased
- Exploitation
  - Uses known information to pick action
  - Learned policy
- We need exploration to learn, to perform well we need exploitation
- How do we reconcile:
  - Offline learning: separate learning and exploitation phases
  - Online learning: ε-greedy policy, decaying exploration

Repeat

    Pick (s,a)
    Make the transition to s'
    Receive reward r
    $Q(s,a) \leftarrow (1-\alpha)Q(s, a) + \alpha(r + \gamma \max Q(s', a))$

Until sufficiently happy with performance

Off-policy / on-policy

- (s, a, r, s') can be data observed from demonstration
  - Recorded data
  - Expert
  - Novice
  - Simulator
- Off-policy - Q-learning
- On-policy - SARSA

Repeat

  Pick (s,a)

  Make the transition to s'

  Receive reward r

  $Q(s,a) \leftarrow (1-\alpha)Q(s, a) + \alpha(r + \gamma \max Q(s', a))$

Until sufficiently happy with performance

Q-Learning Pseudo Code

```
a[da] - actions array
s[ds] - states array
Q[ds,da] - Q table
epsilon = 0.3;
gamma = 0.9; % discount factor;
alpha = 0.9; % learning constant

for i=1..ds, j=1..da
    Q(i,j) = 0;
end
while not done
        currentState = agent.getCurrentState();
        is <- find index 'is' s.t. currentState = ds[is]

        % epsilon greedy policy
        ra = rand(0,1);
        if ra < epsilon
            % select random action
            ia = rand(1,da);
        else
            %choose best one
            ia = argmax(Q[is, j]), j=1,..,da
        end
        currentAction = a[ia];

        % apply action to the current state, observe reward
        r = agent.applyAction(currentAction);
        nextState = agent.getCurrentState();
        isn <- find index 'isn' s.t. nextState = ds[isn]

        Q(is,ia) = (1-alpha)Q(is,ia) + alpha(r +
                    gamma * max(Q(isn, jn), jn=1,..,da))
end
```

Model-free vs. model-based

- Model-based
  - Simulator available
  - Used to create (s, a, s’) tuples
- Model-free
  - (s, a, s’) observed
  - Recorded data
  - Expert
  - Novice

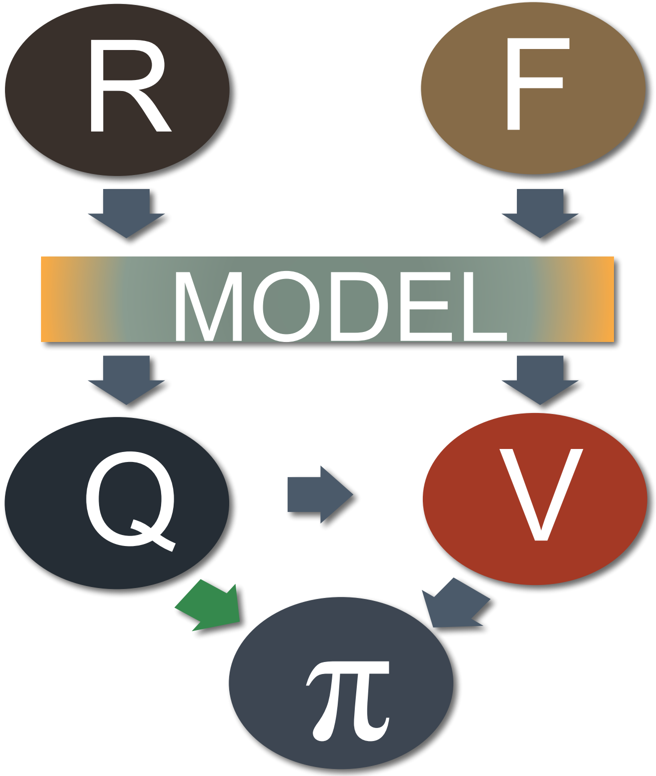
Repeat
    Pick (s,a)
    Make the transition to s’
    Receive reward r
    $Q(s,a) \leftarrow (1-\alpha)Q(s, a) + \alpha(r + \gamma \max_a Q(s', a))$
Until sufficiently happy with performance

Q-Learning Pros and Cons

- Pros
  - Simple
  - Each iteration is fast
  - Extensible framework
- Cons
  - Takes many steps to learn
  - Doesn’t scale up
  - Generalization is hard: rote memory with a bit of bookkeeping
  - Exploration might be unsafe

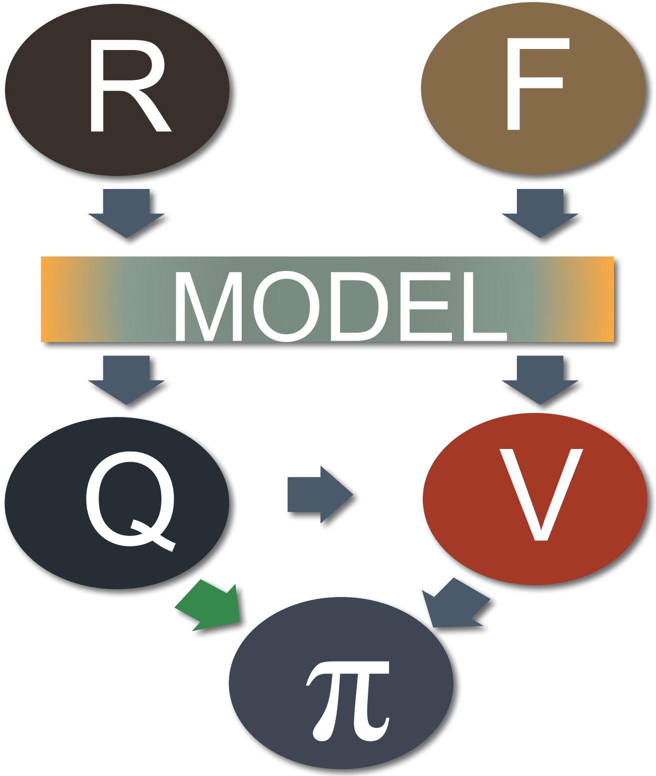
Components of RL

- Model (MDP)
  - Transitions between states and actions
  - Reward, R
- Value functions
  - State value, V: what's the potential payoff for the state
  - State action value, Q: what's the potential payoff for the action
  - Difference between V and Q: going to college example
- Policy, π
  - Where should I go next?



Components of RL

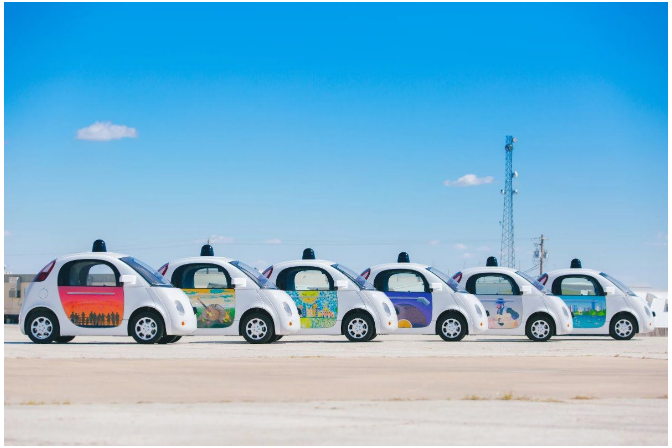
- Model, Value functions, and Policy are related
- Choice of what to learn
  - Model – **model-based RL**
  - Value functions – **value iteration RL**
  - Policy – **policy search**, or **policy iteration**
  - Value and policy - **actor critic methods.**
- Q-Learning is value iteration RL



RL in Continuous Spaces

RL in Continuous Spaces

- 7 DoF robotic arm, each joint 10 position
  - 10 million states, without considering velocity
- What is the state space for the self-driving car?
- What do we do if state space is large? Or continuous?
- Function approximation
  - Parametric, non-parametric
- What functions?
  - Model, Value, Policy
- Much, much faster. Why?
  - Similar states, behave similarly
  - More generalization
- Requires expert knowledge



RL Approximation

- State space infinite
  - Q cannot be tabular
- Action space infinite
  - Action selection approximation
  - Maximization problem
- Generalization
- Compact representation of a function

Repeat

Pick (s,a)

Make the transition to s'

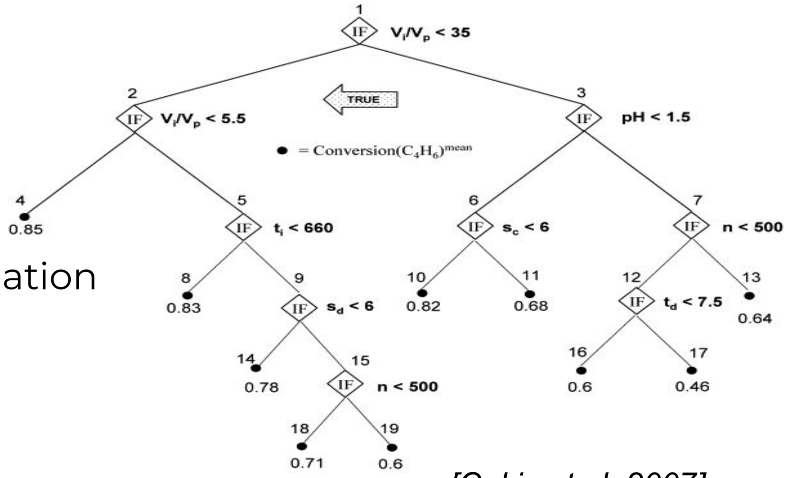
Receive reward r

$Q(s,a) \leftarrow (1-\alpha)Q(s, a) + \alpha(r + \gamma\, \mathbf{max\ Q(s', a)})$

Until sufficiently happy with performance

Approximation Taxonomy

- Parametric $Q_n = F(\theta_n)$
  - Fixed number of parameters
  - Feature vector, F, derived from the problem domain
  - (Deep) neural net
  - Example: $$F(\theta_1, \theta_2, \theta_3) = \theta_1(x-1)^2 + \theta_2(y-2)^2 + \theta_3 xy$$
- Nonparametric
  - Derived from data
  - Data dependent parameters and/or features
  - Examples: regression tree
- Approximation error
  - Difference between true function and approximation
- Convergence, stability, and consistency



[Cukic et al. 2007]

Parametric approximation

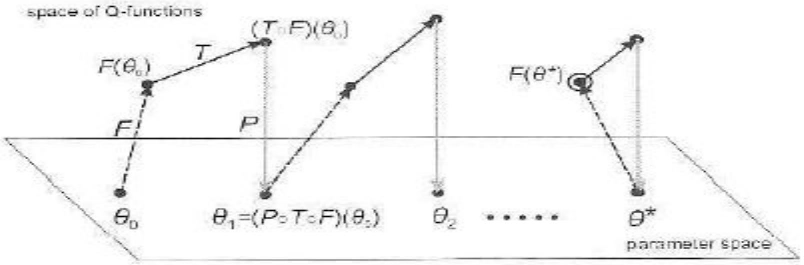
- Linear
  - Feature vector $F(\theta_1, \theta_2, \theta_3) = \theta_1(x-1)^2 + \theta_2(y-2)^2 + \theta_3 xy$
- Nonlinear
  - $F(\theta_1, \theta_2, \theta_3) = \theta_1\theta_2(x-1) + \theta_2(y-2) + \theta_3^2 xy$
- Examples
  - State aggregation
  - Gaussian radial function
  - Problem-specific features
  - Neural networks
- Pros
  - Convergence criteria in linear case
- Cons
  - Difficult convergence criteria in nonlinear case
  - Needs prior knowledge to design features

Nonparametric approximation

- Builds a model from the available data
- Examples
  - Kernels, SVM, regression trees
- Pros
  - Flexible
  - Sample-efficient
- Cons
  - Very problematic convergence criteria
  - Too much data – too complex
  - Overfitting

Approximate Q-Iteration

- Discrete case
  - Update Q function
  - Q(s,a) <- (1-α)Q(s, a) + α(r + γ max Q(s', a))



Businiu et al. 2010

- Approximate case
  - Approximate $Q_n(s,a) = F(\theta_n, s', a)$
  - Update
    - $q_{n+1}(s,a) \leftarrow (1-\alpha)F(\theta_n, s, a) + \alpha(r + \gamma \max_a F(\theta_n, s', a)$
  - Find new approximation / projection
    - $\theta_{n+1} = \text{argmin}_\theta || q_{n+1}(s,a) - F(\theta_n, s, a)||^2$

Stopping Criteria

- Fixed number of iterations
- Convergence $||\theta_{n+1} - \theta_n|| < \epsilon$

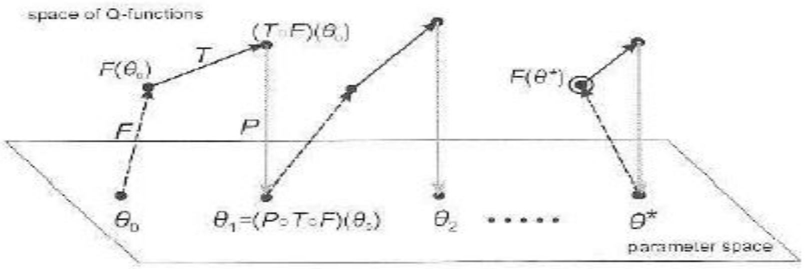
Convergence

- Convergence to optimal policy
  - When approximation is contraction
  - Case to case basis on dynamics, feature vectors, and projections
- Probabilistic bounds on suboptimality
  - Based on number of samples
  - How far off will I be given n samples?

Remi Munos, DeepMind

Approximate Q-iteration Summary

- Q-learning → Approximate Q-iteration
- Three steps for algorithm approximation
  - Function approximation
  - Estimate of new function
  - Projection to the function approximation domain
- Pros
  - Generalization
  - Speed
- Cons
  - Convergence
  - Harder setup



Offline least squares approximate Q-iteration with linear parameterization

- Input
  - Discount factor γ
  - Feature vector $F : S \times A -> \mathbb{R}^n$
  - Samples $\{(s_{l_n}, a_{l_n}, s'_{l_n}, r_{l_n}) | l = 1, ..., m_n, n = 1, .., N\}$
- Start with arbitrary θ_0
- While not done
  - For $l = 1, ..., m$
    - New sample values $Q_{l_{n+1}} = r_{l_n} + \gamma \max_{a \in A} \theta_n^T F(s'_{l_n}, a)$

  $$\theta_{n+1} = \text{argmax}_\theta ||Q_{l_{n+1}} - \theta^T F(s_{l_n}, a_{l_n})||^2$$

  - Find new θ_{n+1} through least squares

Offline non-parametric approximate Q-iteration

- Input
  - Discount factor γ
  - Feature vector $F : S \times A -> \mathbb{R}^n$
  - Samples $\{(s_{l_n}, a_{l_n}, s'_{l_n}, r_{l_n}) | l = 1, ..., m_n, n = 1, .., N\}$
- Start with arbitrary θ_0
- While not done $l = 1, ..., m$
  - For $Q_{l_{n+1}} = r_{l_n} + \gamma \max_{a \in A} \tilde{Q}_n(s'_{l_n}, a)$
    - New sample values
  - Find new $\tilde{Q}_{n+1}$ by solving machine learning problem

    $(s_{l_n}, a_{l_n}, Q_{l_{n+1}})$

Deep Q-learning

- Deep neural network approximates Q function.
- Learn Q's weights.
- Bellman update + supervised learning.
- Add **replay buffer** to create mini-batches.

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode $= 1, M$ **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1$,T **do**

With probability ε select a random action a_t

otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

$$\text{Set } y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$$

Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

End For

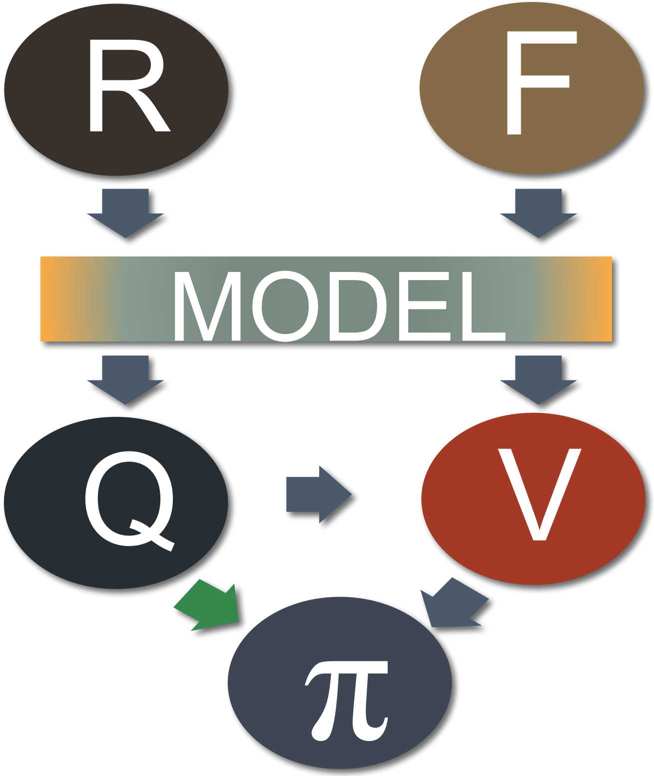
End For

DQN Pseudo Code (https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf)

[redacted]

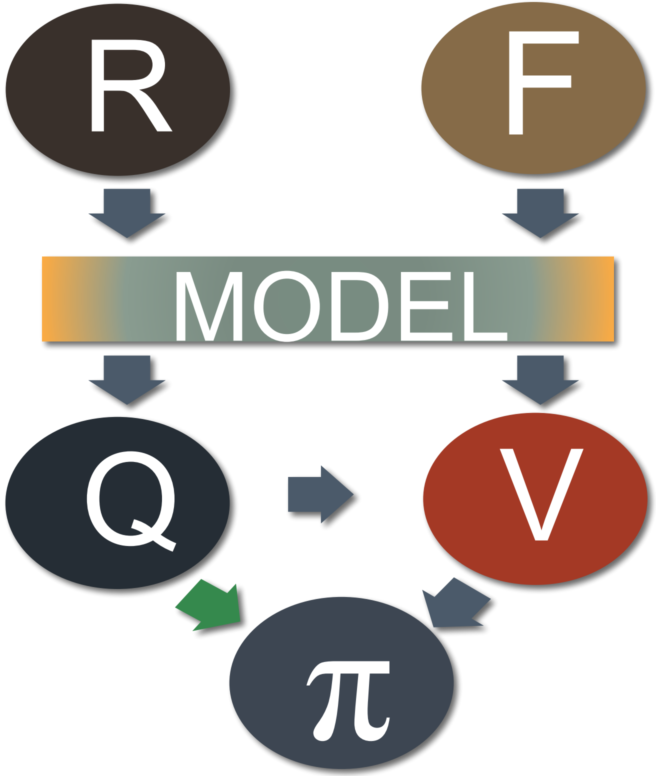
Recap

- RL
  - Method for solving MDP when model is not known.
  - Appropriate for agents interacting with the world trying to accomplish a task.
  - Policy: a sequence of actions that maximize accumulated reward.
- Components of RL problem
  - Model (States, actions, transitions, reward), value functions, policy.
  - Learn any of them. Policy can be derived.
- Small discrete spaces
  - Tabular representation of values
  - Iterations: new estimate = experienced reward + old estimate
- Large spaces
  - Approximation.



Why use RL?

- Trial and error learning
- Learns expectation over cumulative label



Questions

1. What is off-policy learning?
2. What is off-line learning?
3. What is model-free learning?
4. What is deep RL?
5. What does policy search mean?
6. When is exploration important?
7. When is exploitation needed?
8. DDPG is an actor-critic, model-free, off-policy algorithm with DNNs as approximators.
   a. What does that tell you?
9. Kernel based Q-learning.
   a. What does it mean?
10. PPO is an on-policy policy optimization with DNNs.
    a. What does it mean?

Useful links

Reinforcement Learning 101

Dave Silver's class

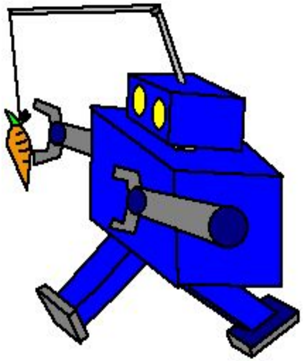
Abbeel and Schulman's tutorial

Introduction to Various Reinforcement Learning Algorithms. Part I (Q-Learning, SARSA, DQN, DDPG)

RL algorithms taxonomy

Thank you

Questions?



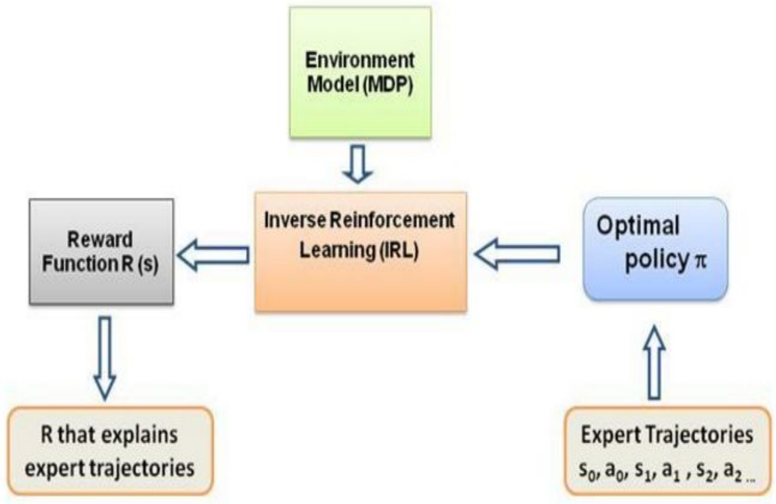
RL and learning from demonstration

Add a slide on off policy, on-policy

- PPO is on-policy RL algorithm, with neural net approximator.
- DDPG is off-policy RL, with neural net approximator

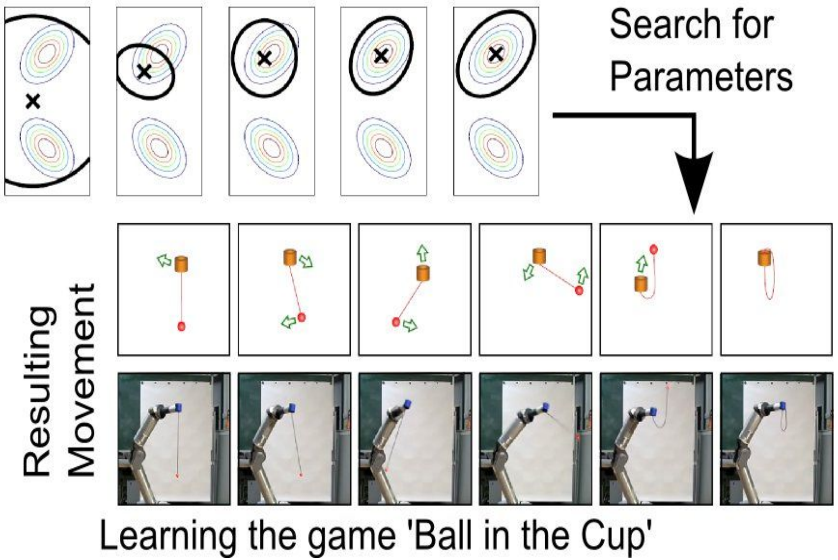
Inverse Reinforcement Learning

- When no reward available
- Expert demonstration available
- Learn reward from the expert demonstration
- Introduction
  - http://www.youtube.com/watch?v=M-QUkgk3HyE
- Peter Abbeel's page
  - http://ai.stanford.edu/~pabbeel/RL-v ideos.html



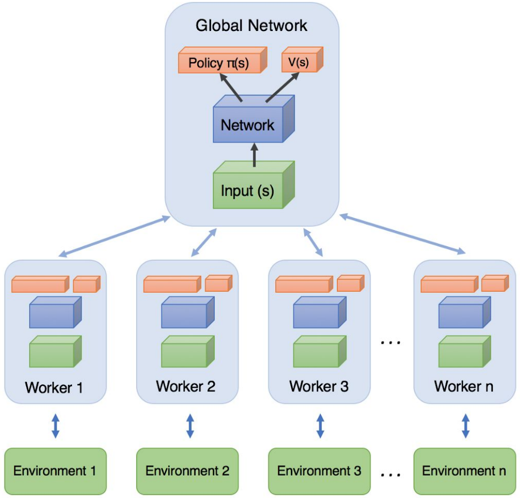
Skill Learning with Policy search

- Given basic motions (motion primitives)
- Policy search
  - Natural actor-critic
  - Weighted combination of the motion primitives
- Batting
  - http://www.youtube.com/watch?v=cPIGGKBnUZI
- Pancake flipping
  - http://www.youtube.com/watch?v=W_gxLKSsSIE



Intro to Deep RL

- DQN
  - Off-policy
  - Q-function - deep convolutional net
  - Experience replay - model-based
- Asynchronous Advantage Actor-Critic (A3C)
  - Actor-critic
  - Advantage: A = R - V(s)
- Thanks to Brandyn:



https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0#.terdpa6v7