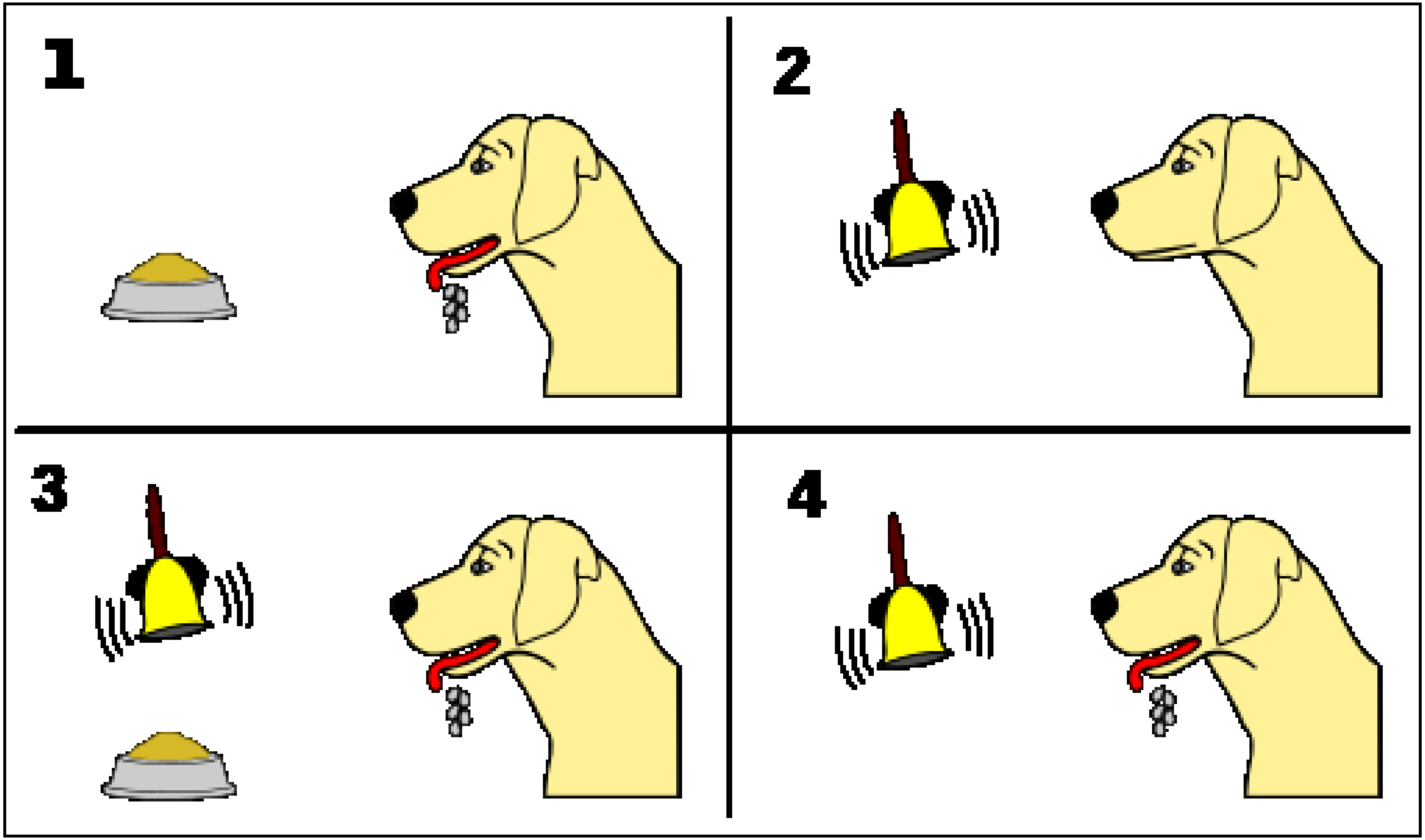
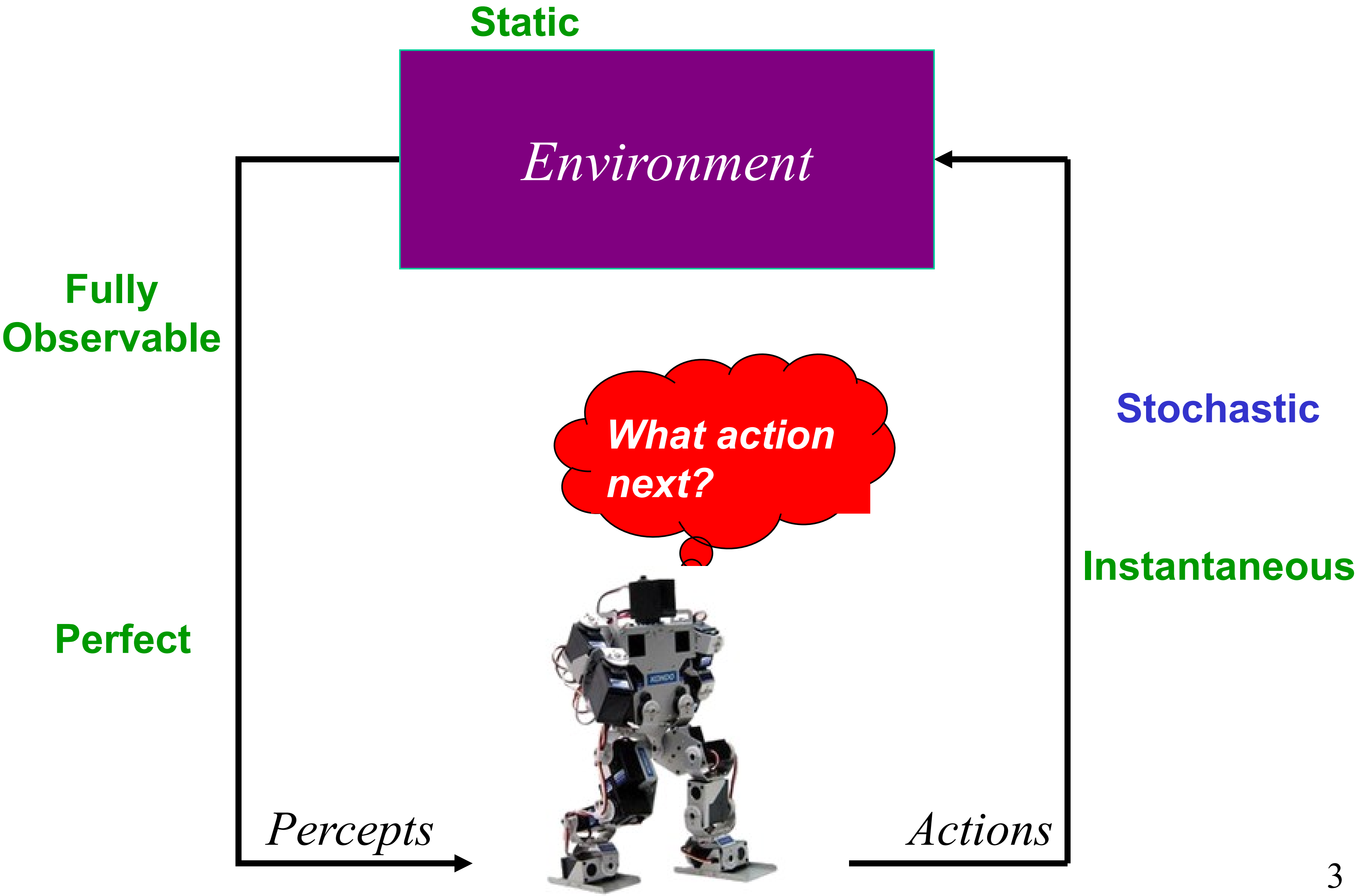
Reinforcement Learning
Chapter 21

Mausam
(some slides by Rajesh Rao)


1
2
3
4

MDPs



Static
Environment
Fully Observable
Perfect
What action next?
Stochastic
Instantaneous
Percepts
Actions

Reinforcement Learning

- S: a set of states
- A: a set of actions
- T(s,a,s'): transition model
- R(s,a): reward model
- γ: discount factor
- Still looking for policy $\pi(s)$

- New Twist: we don't know T and/or R
  - we don't know which state is good/what actions do
  - must learn from data/experience

- Fundamental model for learning of human behavior

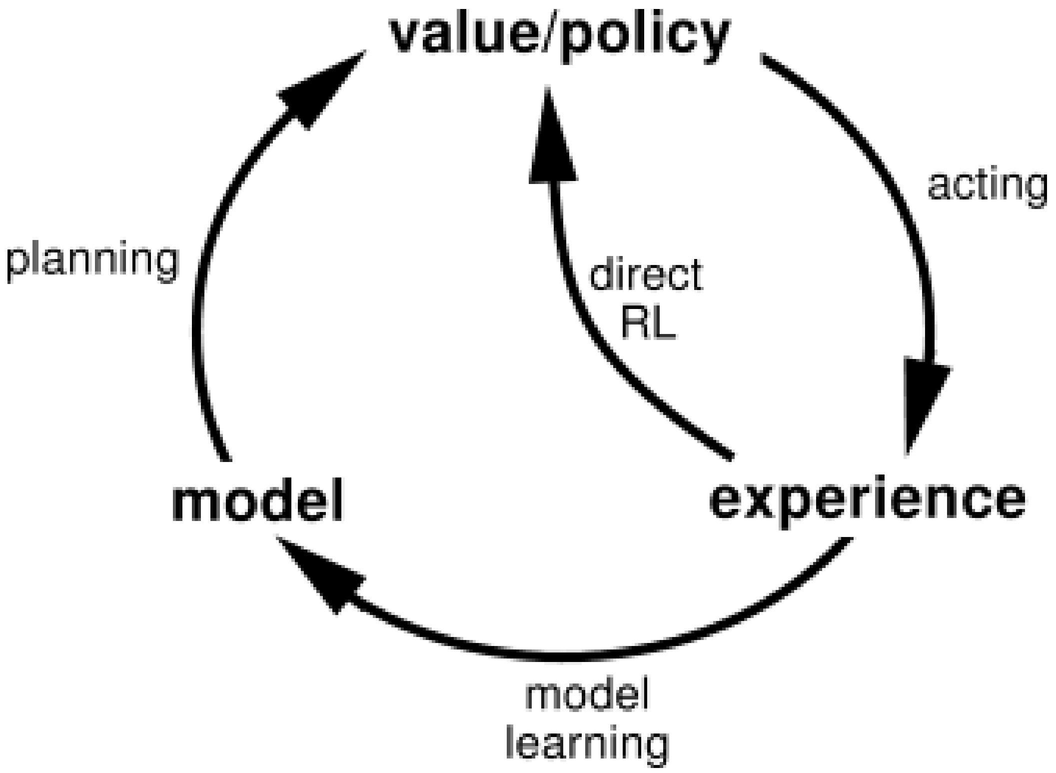
Learning vs Inference

- Batch setting in Bayes Nets
  - Data → Model → Prediction

- Active setting in MDPs
  - Action → Data → (Model?)

  - Actions have two purposes
    - To maximize reward
    - To learn the model

Learning/Planning/Acting



Main Dimensions

- Model-based vs. Model-free
  - Model-based: learn the model (T, R)
  - Model-free: directly learn what action to do when
- Passive vs. Active
  - Passive: learn state values evaluating a given policy
  - Active: need to learn both optimal policy + state values
- Strong vs Weak simulator
  - Strong: can jump to any part of state space and simulate
  - Weak: real world; can't teleport

RL and Animal Foraging

- RL studied experimentally for more than 80 years in psychology and brain science
  - Rewards: food, pain, hunger, drugs, etc.
  - Evidence for RL in the brain via a chemical called dopamine

- Example: foraging
  - Bees can learn near-optimal foraging policy in field of artificial flowers with controlled nectar supplies

Passive Learning (Policy Evaluation)

- Given a policy π: compute V^π
  - V^π : expected discounted reward while following π
- Remember
  - We don't know T
  - We don't know R
  - But we can execute (and simulate)
- Key Idea
  - compute expectations by average over samples

Aside: Expected Age

Goal: Compute expected age of COL333 students

Known P(A)

[redacted] [redacted]

Without P(A), instead collect samples [a_1, a_2, … a_N]

Unknown P(A): "Model Based"

[redacted]

[redacted]

Why does this work? Because eventually you learn the right model.

Unknown P(A): "Model Free"

[redacted]

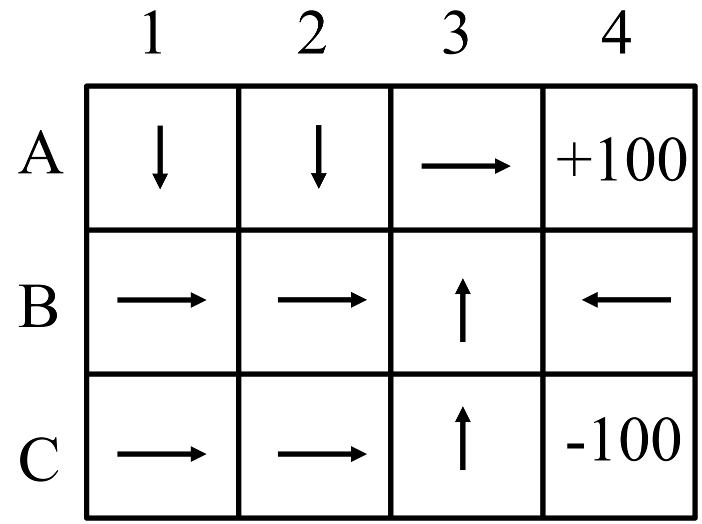
Why does this work? Because samples appear with the right frequencies.

Method 1: Model-based Learning

- Learn an empirical model
- Solve for V^{π} using policy evaluation
  - assuming that the learned model is correct

- Learning the model
  - maintain estimates of T(s,a,s')
  - maintain estimates of R(s,a,s')

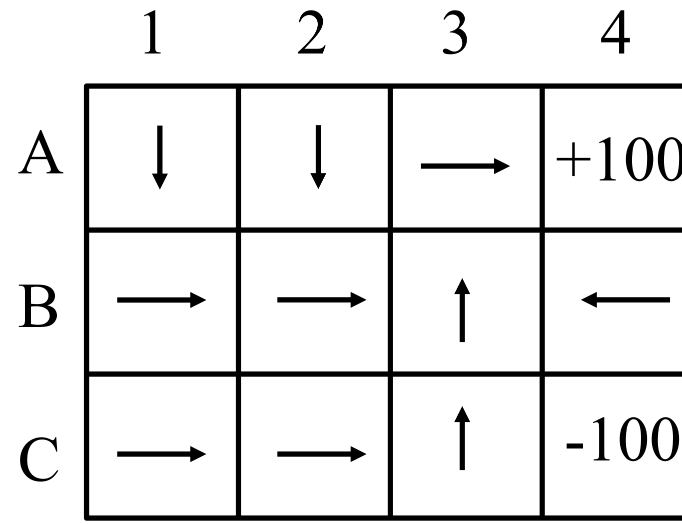
Example



- 12 states, 4 actions
- Reward(action) = -1
- Discount factor = 1
- A4 and C4 are absorbing states

- When might this be the optimal policy?

Data on Executing π



(A1, D, -1)
(B1, R, -1)
(B2, R, -1)
(B3, U, -1)
(A3, R, -1)
(A2, D, -1)
(B2, R, -1)
(B3, U, -1)
(A3, R, -1)
(A4, 100)

(A1, D, -1)
(B1, R, -1)
(B2, R, -1)
(B3, U, -1)
(C3, U, -1)
(C4, -100)

- T(A1, D, B1) = 1
- T(B3, U, A3) = 2/3
- We may want to smooth…

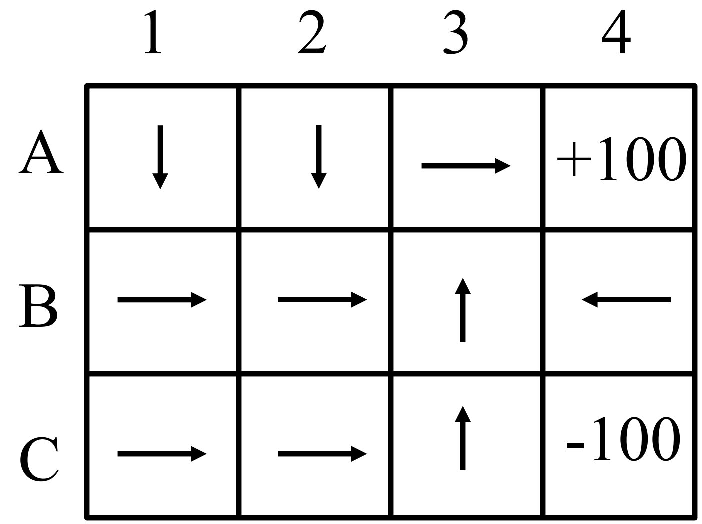
Properties

- Converges to correct model with infinite data
  - If no state is starved
- With correct model
  - V^{π} is computed accurately
- How about model free learning?
  - i.e., expectation is average of samples

Method 2: Empirical Estimation of V^π

- Given a policy π: compute V^π
  - V^π : expected discounted long-term reward following π
  - $V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[long\ term\ reward\ with\ s \rightarrow s']$
  - $V^\pi(s) = \frac{1}{N} \sum_i\ [long\ term\ reward_i]$

Data on Executing π



(A1, D, -1)
(B1, R, -1)
(B2, R, -1)
(B3, U, -1)
(A3, R, -1)
(A2, D, -1)
(B2, R, -1)
(B3, U, -1)
(A3, R, -1)
(A4, 100)

(A1, D, -1)
(B1, R, -1)
(B2, R, -1)
(B3, U, -1)
(C3, U, -1)
(C4, -100)

- V^{π} (B1) =
- V^{π} (B2) =

Properties

- Converges to optimal with infinite data
  - If no state is starved
- Is wasteful (why?)
  - Compare V^{π} (B1) and V^{π} (B2)
- Each state is computed independently
  - Connections (Bellman equations) are ignored
  - Learns slowly

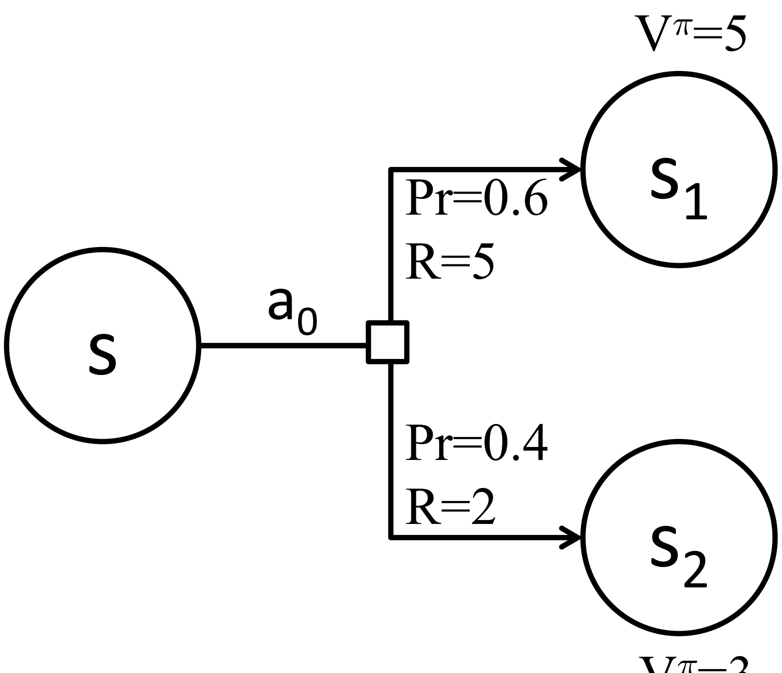
Method 3: Temporal Difference Learning

- Given a policy π: compute V^π
  - V^π : expected discounted long-term reward following π
  - $V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[long\ term\ reward\ with\ s \rightarrow s']$
  - $V^\pi(s) = \frac{1}{N}\sum_i\ [long\ term\ reward_i]$

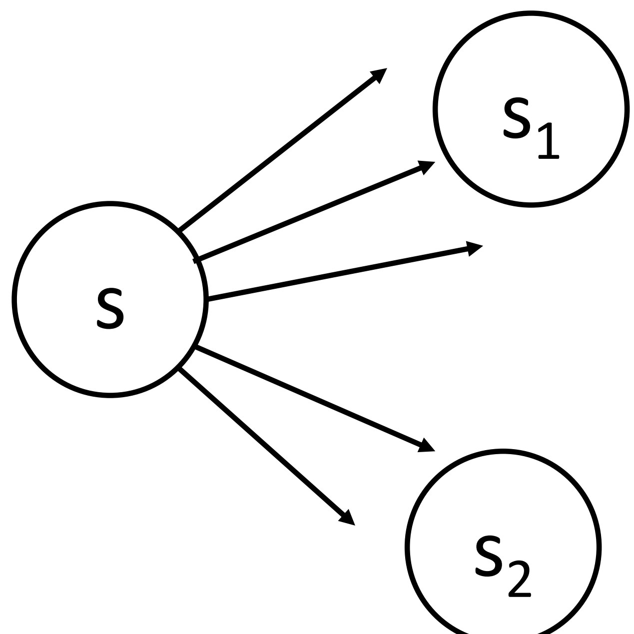
- $V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')]$
- represents relationship between s and s'
- TD Learning: computing this expectation as average

TD Learning

- $V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$
- Say I know correct values of $V^{\pi}(s_1)$ and $V^{\pi}(s_2)$



V^{π}(s)=0.6(5+5) + 0.4(2+3)
= 6 + 2 = 8



V^{π}(s)= (10+10+10+5+5)/5
= 8

TD Learning

- $V^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^\pi(s')]$
- Inner term is the sample value
  - (s,s',r): reached s' from s by executing $\pi(s)$ and got immediate reward of r
  - sample = $\mathrm{r} + \gamma V^\pi(s')$
- Compute $V^\pi(s) = \frac{1}{N}\sum_i sample_i$

- Problem: we don't know true values of $V^\pi(s')$
  - learn together using dynamic programming!

Estimating mean via online updates

- Don't learn T or R; directly maintain V^{π}
- Update $V^{\pi}(s)$ each time you take an action in s via a moving average
  - $V^{\pi}_{n+1}(s) \leftarrow \frac{1}{n+1}(n.\ V^{\pi}_{n}(s) + \text{sample}_{n+1})$
  - $V^{\pi}_{n+1}(s) \leftarrow \frac{1}{n+1}((n+1-1).V^{\pi}_{n}(s) + \text{sample}_{n+1})$
  - $V^{\pi}_{n+1}(s) \leftarrow V^{\pi}_{n}(s) + \frac{1}{n+1}(\text{sample}_{n+1} - V^{\pi}_{n}(s))$

average of n+1 samples

learning rate

sample n+1

  - $V^{\pi}_{n+1}(s) \leftarrow V^{\pi}_{n}(s) + \alpha(\text{sample}_{n+1} - V^{\pi}_{n}(s))$
  - Nudge the old estimate towards the sample

TD Learning

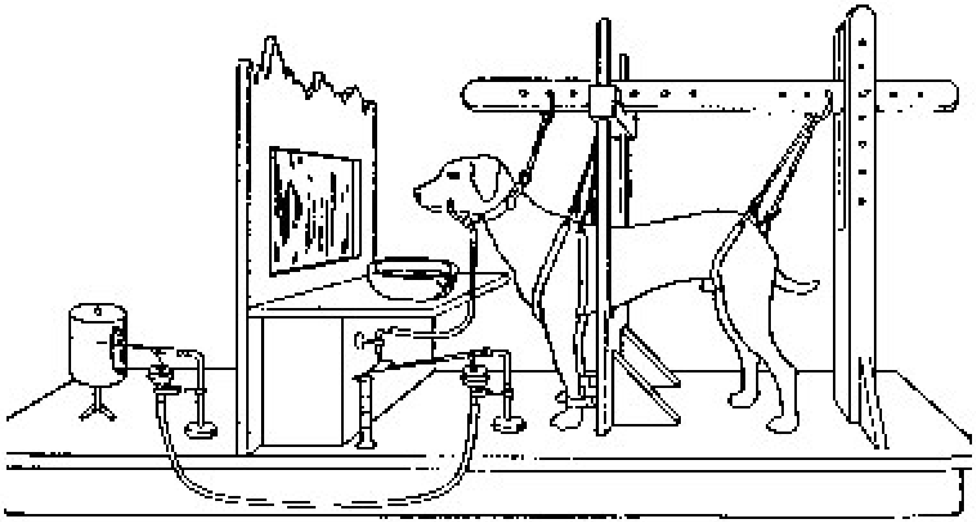
- (s,s',r)
- $V^{\pi}(\text{s}) \leftarrow V^{\pi}(\text{s}) + \alpha(\text{sample} - V^{\pi}(\text{s}))$
- $V^{\pi}(\text{s}) \leftarrow V^{\pi}(\text{s}) + \alpha(\text{r} + \gamma V^{\pi}(s') - V^{\pi}(\text{s}))$ TD-error
- $V^{\pi}(\text{s}) \leftarrow (1-\alpha)V^{\pi}(\text{s}) + \alpha(\text{r} + \gamma V^{\pi}(s'))$

- Update maintains a mean of (noisy) value samples

- If the learning rate decreases appropriately with the number of samples (e.g. 1/n) then the value estimates will converge to true values! (non-trivial)

Early Results: Pavlov and his Dog

- Classical (Pavlovian) conditioning experiments
- Training: Bell →Food
- After: Bell → Salivate
- Conditioned stimulus (bell) predicts future reward (food)



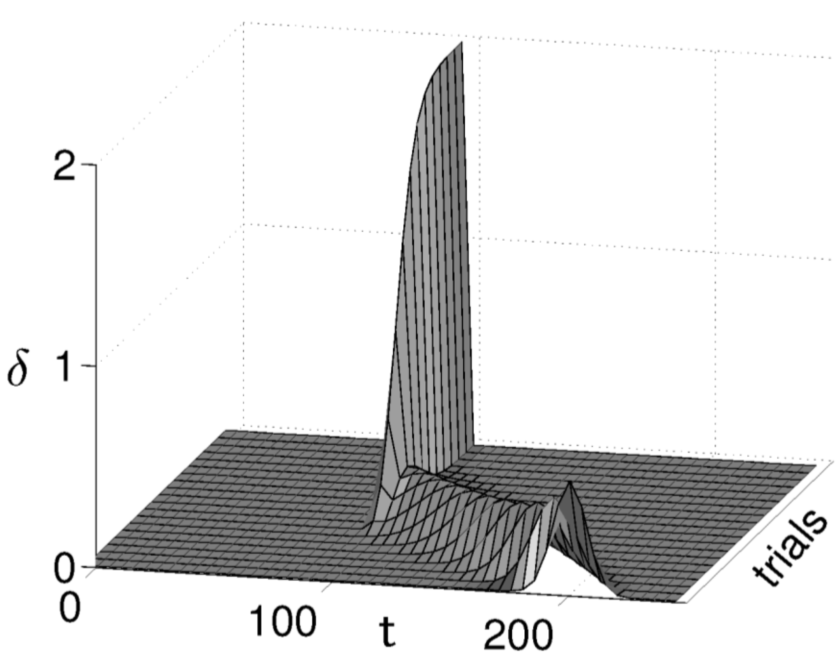
Predicting Delayed Rewards

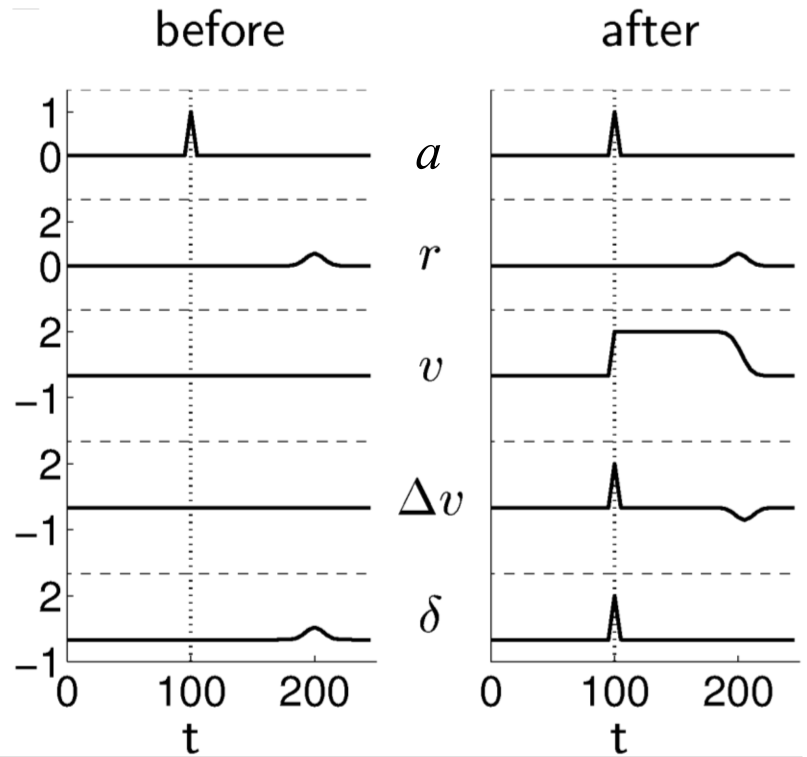
- Reward is typically delivered at the end (when you know whether you succeeded or not)
- Time: $0 \leq t \leq T$ with stimulus a(t) and reward r(t) at each time step t (Note: r(t) can be zero at some time points)
- Key Idea: Make the output v(t) predict total expected future reward starting from time t

$$v(t) \approx \left\langle \sum_{\tau=0}^{T-t} r(t+\tau) \right\rangle$$

Predicting Delayed Reward: TD Learning

Stimulus at t = 100 and reward at t = 200





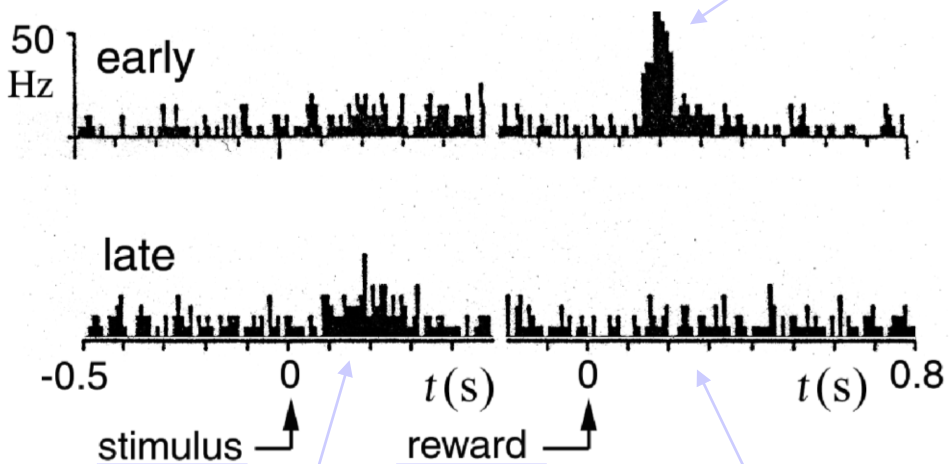
Prediction error δ for each time step
(over many trials)

Figure from Theoretical Neuroscience by Peter Dayan and Larry Abbott, MIT Press, 2001

Prediction Error in the Primate Brain?

Dopaminergic cells in Ventral Tegmental Area (VTA)

Reward Prediction error? $[r(t) + v(t+1) - v(t)]$



Before Training

After Training

$[0 + v(t+1) - v(t)]$

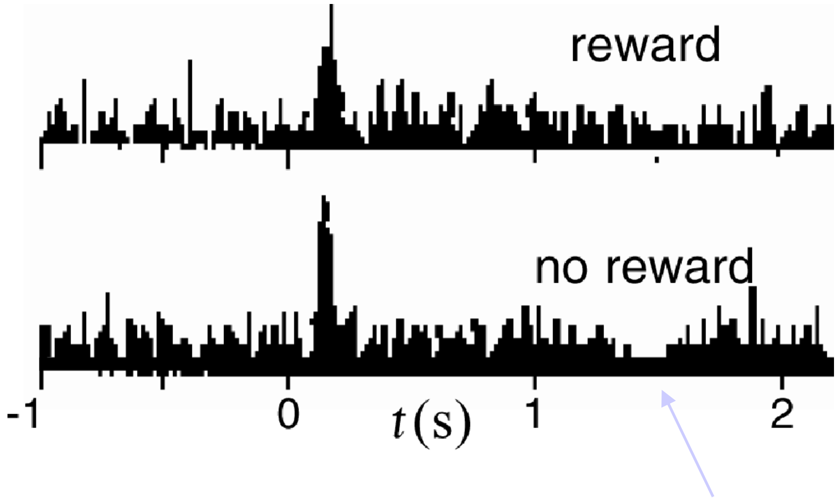
No error

$v(t) \approx r(t) + v(t+1)$

Figure from Theoretical Neuroscience by Peter Dayan and Larry Abbott, MIT Press, 2001

More Evidence for Prediction Error Signals

Dopaminergic cells in VTA



Negative error

$$r(t) = 0, v(t+1) = 0$$

$$[r(t) + v(t+1) - v(t)] = -v(t)$$

Figure from Theoretical Neuroscience by Peter Dayan and Larry Abbott, MIT Press, 2001

The Story So Far: MDPs and RL

Known MDP: Offline Solution

Goal	Technique
Compute V*, Q*, π*	Value / policy iteration
Evaluate a policy π	Policy evaluation

Unknown MDP: Model-Based

Goal	Technique
Compute V*, Q*, π*	VI/PI on approx. MDP
Evaluate a policy π	PE on approx. MDP

Unknown MDP: Model-Free

Goal	Technique
Compute V*, Q*, π*	Q-learning
Evaluate a policy π	TD-Learning

Model-based RL

- Learn an initial model M_0
- Loop
  - VI/PI on M_i to compute policy π_i
  - Execute π_i to generate data
  - Learn a better model M_{i+1}

- Key challenge?

Model-based RL Example

	1	2	3	4
A	↓	?	?	+100
B	↓	?	?	?
C	→	→	→	-2

- Say world is deterministic
  - and no wind

- Lets say the agent first discovers the path to bad reward first

- Will the agent ever learn the optimal policy?
  - won't have any information about some states or state-action pairs

Model-based RL

- Learn an initial model M_0
- Loop
  - VI/PI on M_i to compute policy π_i
  - Execute π_i to generate data
  - Learn a better model M_{i+1}

- Key challenge
  - Just executing π_i is not enough!
  - It may miss important regions
  - Needs to explore new regions

TD Learning → TD (V*) Learning

- Can we do TD-like updates on V*?

- $V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$

- Hmmm… what to do?
  - RHS should be expectation.
  - Instead of V* write all equations in Q*

Bellman Equations (V*) → Bellman Equations (Q*)

- $V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$

- $Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$

- $Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s' + \gamma \max_{a'} Q^*(s',a')]$

- VI → Q-Value Iteration
- TD Learning → Q Learning

Q Learning

- Directly learn Q*(s,a) values
- Receive a sample (s, a, s', r)
- Your old estimate Q(s,a)
- New sample value: r+$\gamma \max_{a'} Q(s', a')$

Nudge the estimates:

- Q(s,a) ← Q(s,a) + α(r+$\gamma \max_{a'} Q(s', a') - Q$(s,a))
- Q(s,a) ← $(1-\alpha)Q$(s,a)+ α(r+$\gamma \max_{a'} Q(s', a')$)

Q Learning Algorithm

- **Forall s, a**
  - **Initialize Q(s, a) = 0**

- **Repeat Forever**

  Where are you? s.

  Choose some action a

  Execute it in real world: *(s, a, r, s')*

  Do update:

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

Is an *off policy learning* algorithm

Properties

- Q Learning converges to optimal values Q*
  - Irrespective of initialization,
  - Irrespective of action choice policy
  - Irrespective of learning rate

- as long as
  - states/actions finite, all rewards bounded
  - No (s,a) is starved: infinite visits over infinite samples
  - Learning rate decays with visits to state-action pairs
    - but not too fast decay. ($\sum_i \alpha(s,a,i) = \infty$, $\sum_i \alpha^2(s,a,i) < \infty$)

Q Learning Algorithm

- **Forall s, a**
  - Initialize Q(s, a) = 0
- **Repeat Forever**

  Where are you? s.

  Choose some action a

  Execute it in real world: *(s, a, r, s')*

  Do update:

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

How to choose?

new: exploration

greedy: exploitation

Exploration vs. Exploitation Tradeoff

- A fundamental tradeoff in RL
- Exploration: must take actions that may be suboptimal but help discover new rewards and in the long run increase utility
- Exploitation: must take actions that are known to be good (and seem currently optimal) to optimize the overall utility
- Slowly move from exploration → exploitation

Explore/Exploit Policies

- Simplest scheme: ϵ-greedy
  - Every time step flip a coin
  - With probability 1-ϵ, take the greedy action
  - With probability ϵ, take a random action

- Problem
  - Exploration probability is constant

- Solutions
  - Lower ϵ over time
  - Use an exploration function

Explore/Exploit Policies

- Boltzmann Exploration
  - Select action a with probability
  - $\Pr(a|s) = \frac{\exp(Q(s,a)/T)}{\sum_{a' \in A} \exp(Q(s,a')/T)}$
- T: Temperature
  - Similar to simulated annealing
  - Large T: uniform, Small T: greedy
  - Start with large T and decrease with time
- GLIE: greedy in the limit of infinite exploration

Explore/Exploit Policies

- Exploration Functions
  - stop exploring actions whose badness is established
  - continue exploring other actions
- Let Q(s,a) = q, #visits(s,a) = n
- E.g.: $\mathrm{f}(\mathrm{q}, \mathrm{n}) = q + k/n$
  - Unexplored states have infinite f
  - Highly explored bad states have low f
- Modified Q update
  - Q(s,a) ← $(1-\alpha)Q$(s,a)

$$+ \alpha(r + \gamma \max_{a'} f(Q(s', a'), N(s', a')))$$

States leading to unexplored states are also preferred

Explore/Exploit Policies

- A Famous Exploration Policy: UCB
  - Upper Confidence Bound

$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

Value Term:
favors actions that looked good historically

Exploration Term:
actions get an exploration bonus that grows with ln(n)

Optimistic in the Face of Uncertainty

Model based vs. Model Free RL

- Model based
  - estimate $O(|\mathcal{S}|^2|\mathcal{A}|)$ parameters
  - requires relatively larger data for learning
  - can make use of background knowledge easily

- Model free
  - estimate $O(|\mathcal{S}||\mathcal{A}|)$ parameters
  - requires relatively less data for learning

Generalizing Across States

- Basic Q-Learning (or VI) keeps a table of all q-values

- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning

Feature-based Representation

- Describe a state using vector of features
- We can write a q function using a few weights:

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few powerful numbers (w_i)

- Disadvantage: states may share features but actually be very different in value!

Approximate Q-Learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Exact Q-Learning
  - $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$

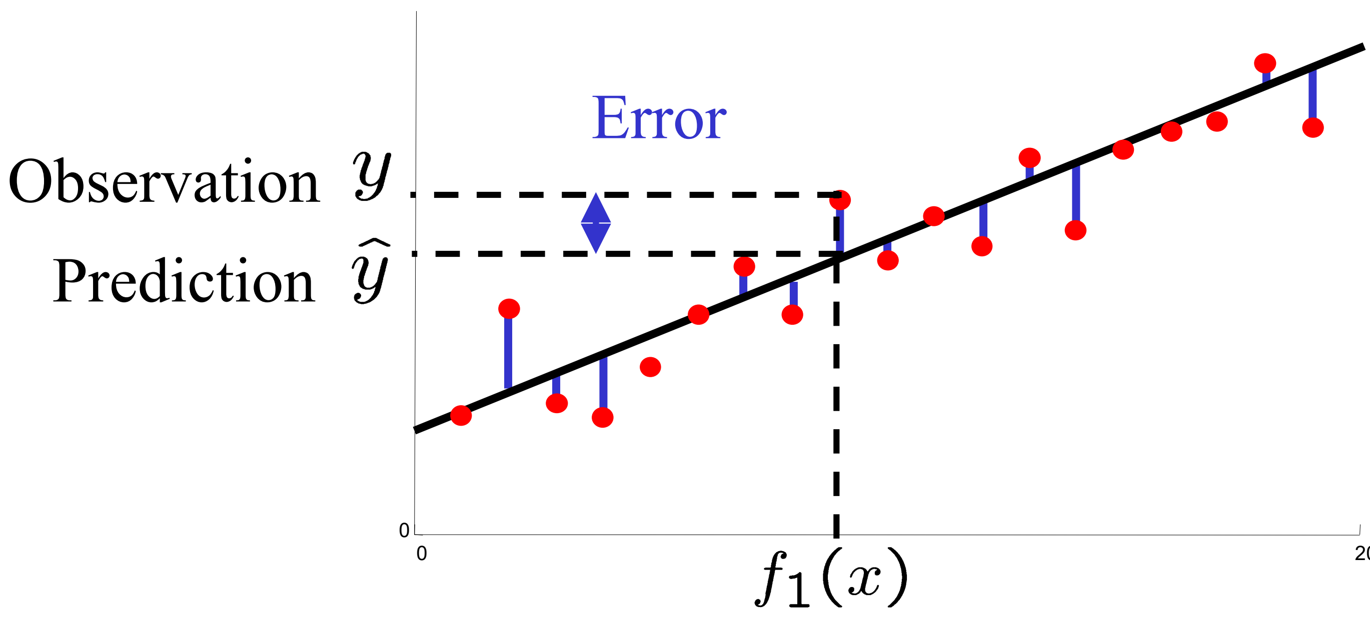
  difference

- Q-Learning with linear function approximation
  - $w_m \leftarrow w_m + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) f_m(s,a)$

- Move feature weights up/down based on difference and feature values

Optimization: Least Squares

$$\text{total error} = \sum_i (y_i - \widehat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x_i) \right)^2$$



Minimizing Error

Imagine we had only one point x, with features f(x), target value y, and weights w:

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\ \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

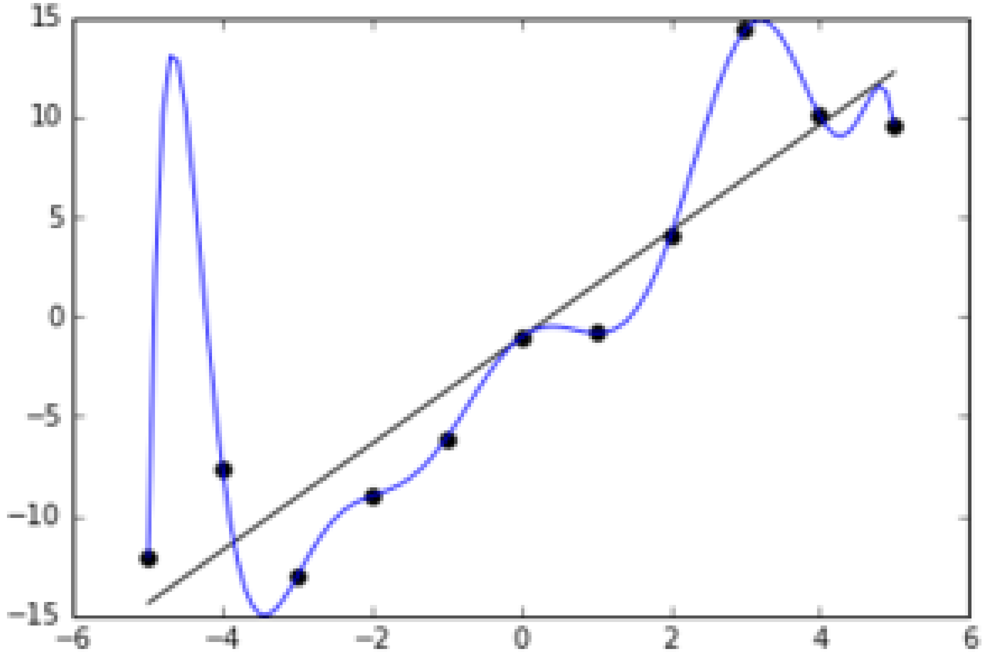
$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

Approximate q update explained:

$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$$

"target" "prediction"

Overfitting and Limited Capacity Approximations



Low capacity generalizes better

Issue: linear approximation not powerful enough in practice

  Deep Learning!

Summary: RL

RL is a very general AI problem

most general single agent?

Main idea: expectation_P as avg of samples

sampling distribution is P

Agent learns as it gathers experience

Exploration-exploitation tradeoff

Function approximation is key: deep RL is the rage!

Applications

- Stochastic Games
- Robotics: navigation, helicopter manuevers…
- Finance: options, investments
- Communication Networks
- Medicine: Radiation planning for cancer
- Controlling workflows
- Optimize bidding decisions in auctions
- Traffic flow optimization
- Aircraft queueing for landing; airline meal provisioning
- Optimizing software on mobiles
- Forest firefighting
- …