Overview

- Reinforcement learning: rough overview
  - mainly following Sutton & Barto 1998

- Learning theory
  - classical & instrumental conditioning

- Dopamine
  - prediction errors and more

- Fitting behaviour with RL models
  - some applied tips & tricks
Types of learning

- Supervised
- Unsupervised
- Reinforcement learning
Setup

\[ \{a_t\} \leftarrow \arg\max_{\{a_t\}} \sum_{t=1}^{\infty} r_t \]

After Sutton and Barto 1998
State space

Electric shocks -1

Gold +1
A Markov Decision Problem

\[ s_t \in S \]
\[ a_t \in A \]
\[ T_{ss'}^a = p(s_{t+1} | s_t, a_t) \]
\[ r_t \sim R(s_{t+1}, a_t, s_t) \]
\[ \pi(a | s) = p(a | s) \]
MDP

\[ s_t \in S \]

\[ a_t \in A \]

\[ T_{ss'}^a = p(s_{t+1} \mid s_t, a_t) \]

\[ r_t \sim R(s_{t+1}, a_t, s_t) \]

\[ \pi(a \mid s) = p(a \mid s) \]
Reinforcement learning

MDP

\[ s_t \in S \]
\[ a_t \in A \]
\[ T_{ss'}^a = p(s_{t+1} | s_t, a_t) \]
\[ r_t \sim R(s_{t+1}, a_t, s_t) \]
\[ \pi(a | s) = p(a | s) \]
Actions

Action left

Action right

\[
T_{\text{left}} = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
T_{\text{right}} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1
\end{bmatrix}
\]
Actions

Action left

\[ T^{\text{left}} = \begin{bmatrix}
.8 & .8 & 0 & 0 & 0 & 0 & 0 \\
.2 & .2 & .8 & 0 & 0 & 0 & 0 \\
0 & 0 & .2 & .8 & 0 & 0 & 0 \\
0 & 0 & 0 & .2 & .8 & 0 & 0 \\
0 & 0 & 0 & 0 & .2 & .8 & 0 \\
0 & 0 & 0 & 0 & 0 & .2 & .8 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix} \]

Action right

\[ T^{\text{right}} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix} \]

Noisy: plants, environments, agent
Actions

Action left

\[
T_{\text{left}} = \begin{bmatrix}
.8 & .8 & 0 & 0 & 0 & 0 & 0 \\
.2 & .2 & .8 & 0 & 0 & 0 & 0 \\
0 & 0 & .2 & .8 & 0 & 0 & 0 \\
0 & 0 & 0 & .2 & .8 & 0 & 0 \\
0 & 0 & 0 & 0 & .2 & .8 & 0 \\
0 & 0 & 0 & 0 & 0 & .2 & .8 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

Action right

\[
T_{\text{right}} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 \\
\end{bmatrix}
\]

Noisy: plants, environments, agent

Absorbing state \(\rightarrow\) max eigenvalue \(< 1\)
Markovian dynamics

\[ p(s_{t+1}|a_t, s_t, a_{t-1}, s_{t-1}, a_{t-2}, s_{t-2}, \cdots) = p(s_{t+1}|a_t, s_t) \]
Markovian dynamics

\[ p(s_{t+1} \mid a_t, s_t, a_{t-1}, s_{t-1}, a_{t-2}, s_{t-2}, \cdots) = p(s_{t+1} \mid a_t, s_t) \]

\[ \mathbf{a}_{t-2}, s_{t-2} \rightarrow \mathbf{a}_{t-1}, s_{t-1} \rightarrow \mathbf{a}_t, s_t \]

Velocity
Markovian dynamics

\[ p(s_{t+1} | a_t, s_t, a_{t-1}, s_{t-1}, a_{t-2}, s_{t-2}, \cdots) = p(s_{t+1} | a_t, s_t) \]

Velocity

\[ s' = [\text{position}] \rightarrow s' = [ \text{position} \ \text{velocity} ] \]
\[ s_t \in S \]
\[ a_t \in A \]
\[ T_{s,s'}^a = p(s_{t+1} | s_t, a_t) \]
\[ r_t \sim R(s_{t+1}, a_t, s_t) \]
\[ \pi(a | s) = p(a | s) \]
Reinforcement learning

MDP

\[ s_t \in S \]
\[ a_t \in A \]
\[ T_{ss'}^a = p(s_{t+1} | s_t, a_t) \]
\[ r_t \sim R(s_{t+1}, a_t, s_t) \]
\[ \pi(a | s) = p(a | s) \]

\[ Thursday, 15 August 13 \]
MDP

\[ s_t \in S \]
\[ a_t \in A \]
\[ T_{ss'}^{a} = p(s_{t+1} | s_t, a_t) \]
\[ r_t \sim R(s_{t+1}, a_t, s_t) \]
\[ \pi(a | s) = p(a | s) \]
Tall orders

- Aim: maximise total future reward

\[ \sum_{t=1}^{\infty} r_t \]

- i.e. we have to sum over paths through the future and weigh each by its probability

- Best policy achieves best long-term reward
Exhaustive tree search
Exhaustive tree search

Decision 1
- o1
- a3
- a4
- a3
- a4
- a3
- a4
- a3
- a4
- a3
- a4
- a4

Decision 2
- a1
- o2
- o3
- a1
- o2
- o3
- a1
- o2
- o3
- a1
- o2
- o3
- a1
- o2
- o3

w^d
Decision tree

$$\sum_{t=1}^{\infty} r_t$$
Decision tree

\[ \sum_{t=1}^{\infty} r_t \]
Decision tree

\[ \sum_{t=1}^{\infty} r_t \]

\[
\begin{array}{c}
8 \\
64
\end{array}
\]
Decision tree

\[
\sum_{t=1}^{\infty} r_t
\]
Policy for this talk

- Pose the problem mathematically
- Policy evaluation
- Policy iteration
- Monte Carlo techniques: experience samples
- TD learning

Policy

Evaluate  ➔  Update
Evaluating a policy

- Aim: maximise total future reward
  \[ \sum_{t=1}^{\infty} r_t \]
- To know which is best, evaluate it first
- The policy determines the expected reward from each state

\[ V^\pi(s_1) = \mathbb{E} \left[ \sum_{t=1}^{\infty} r_t | s_1 = 1, a_t \sim \pi \right] \]
Discounting

- Given a policy, each state has an expected value
  \[ V^\pi(s_1) = \mathbb{E} \left[ \sum_{t=1}^{\infty} r_t \mid s_1 = 1, a_t \sim \pi \right] \]

- But:
  \[ \sum_{t=0}^{\infty} r_t = \infty \]

- Episodic
  \[ \sum_{t=0}^{T} r_t < \infty \]

- Discounted
  - infinite horizons
    \[ \sum_{t=0}^{\infty} \gamma^t r_t < \infty \]
  - finite, exponentially distributed horizons
    \[ \sum_{t=0}^{T} \gamma^t r_t \quad T \sim \frac{1}{\tau} e^{t/\tau} \]
Discounting

- Given a policy, each state has an expected value

\[ V^\pi(s_1) = \mathbb{E} \left[ \sum_{t=1}^{\infty} r_t \mid s_1 = 1, a_t \sim \pi \right] \]

- But:

\[ \sum_{t=0}^{\infty} r_t = \infty \]

- Episodic

\[ \sum_{t=0}^{T} r_t < \infty \]

- Discounted
  
  - infinite horizons
    \[ \sum_{t=0}^{\infty} \gamma^t r_t < \infty \]
  
  - finite, exponentially distributed horizons
    \[ \sum_{t=0}^{T} \gamma^t r_t \quad T \sim \frac{1}{\tau} e^{t/\tau} \]
Discounting

- Given a policy, each state has an expected value

\[ V^\pi(s_1) = \mathbb{E} \left[ \sum_{t=1}^{\infty} r_t | s_1 = 1, a_t \sim \pi \right] \]

- But:

\[ \sum_{t=0}^{\infty} r_t = \infty \]

- Episodic

\[ \sum_{t=0}^{T} r_t < \infty \]

- Discounted
  - infinite horizons

\[ \sum_{t=0}^{\infty} \gamma^t r_t < \infty \]

  - finite, exponentially distributed horizons

\[ \sum_{t=0}^{T} \gamma^t r_t \quad T \sim \frac{1}{\tau} e^{t/\tau} \]
**Markov Decision Problems**

\[
V^\pi(s_t) = \mathbb{E} \left[ \sum_{t'=1}^{\infty} r_{t'} | s_t = s, \pi \right]
\]

\[
= \mathbb{E} \left[ r_1 | s_t = s, \pi \right] + \mathbb{E} \left[ \sum_{t=2}^{\infty} r_t | s_t = s, \pi \right]
\]

\[
= \mathbb{E} \left[ r_1 | s_t = s, \pi \right] + \mathbb{E} \left[ V^\pi(s_{t+1}) | s_t = s, \pi \right]
\]

This dynamic consistency is key to many solution approaches. It states that the value of a state \(s\) is related to the values of its successor states \(s'\).
Markov Decision Problems

\[ V^\pi(s_t) = \mathbb{E}[r_1 | s_t = s, \pi] + \mathbb{E}[V(s_{t+1}), \pi] \]

\[ r_1 \sim \mathcal{R}(s_2, a_1, s_1) \]

\[ \mathbb{E}[r_1 | s_t = s, \pi] = \mathbb{E}\left[ \sum_{s_{t+1}} p(s_{t+1} | s_t, a_t) \mathcal{R}(s_{t+1}, a_t, s_t) \right] \]

\[ = \sum_{a_t} \mathbb{P}(a_t | s_t) \left[ \sum_{s_{t+1}} p(s_{t+1} | s_t, a_t) \mathcal{R}(s_{t+1}, a_t, s_t) \right] \]

\[ = \sum_{a_t} \pi(a_t, s_t) \left[ \sum_{s_{t+1}} \mathcal{T}_{s_t s_{t+1}}^{a_t} \mathcal{R}(s_{t+1}, a_t, s_t) \right] \]
Bellman equation

\[
V^\pi(s_t) = \mathbb{E}[r_1|s_t = s, \pi] + \mathbb{E}[V(s_{t+1}), \pi]
\]

\[
\mathbb{E}[r_1|s_t, \pi] = \sum_a \pi(a, s_t) \left[ \sum_{s_{t+1}} T_{s_t s_{t+1}}^a R(s_{t+1}, a, s_t) \right]
\]

\[
\mathbb{E}[V^\pi(s_{t+1}), \pi, s_t] = \sum_a \pi(a, s_t) \left[ \sum_{s_{t+1}} T_{s_t s_{t+1}}^a V^\pi(s_{t+1}) \right]
\]

\[
V^\pi(s) = \sum_a \pi(a|s) \left[ \sum_{s'} T_{ss'}^a [R(s', a, s) + V^\pi(s')] \right]
\]
Bellman Equation

\[ V^\pi(s) = \sum_a \pi(a|s) \left[ \sum_{s'} T^{a}_{ss'} [R(s', a, s) + V^\pi(s')] \right] \]
Bellman Equation

All future reward from state $s$  

$V^\pi(s) = \sum_a \pi(a|s) \left[ \sum_{s'} T^{a}_{ss'} [R(s', a, s) + V^\pi(s')] \right]$
Bellman Equation

\[ V^\pi(s) = \sum_{a} \pi(a|s) \left[ \sum_{s'} \mathcal{T}^a_{ss'} \left[ R(s', a, s) + V^\pi(s') \right] \right] \]

All future reward from state s

= \quad \mathbb{E} \quad \text{Immediate reward} \quad + \quad \text{All future reward from next state s'}
$Q$ values = state-action values

\[
V^\pi(s) = \sum_a \pi(a|s) \left[ \sum_{s'} T_{ss'}^a \left[ R(s', a, s) + V^\pi(s') \right] \right]
\]

- so we can define state-action values as:

\[
Q(s, a) = \sum_{s'} T_{ss'}^a \left[ R(s', a, s) + V(s') \right]
= \mathbb{E} \left[ \sum_{t=1}^{\infty} r_t | s, a \right]
\]

- and state values are average state-action values:

\[
V(s) = \sum_a \pi(a|s) Q(s, a)
\]
Bellman Equation

\[ V^\pi(s) = \sum_a \pi(a|s) \left[ \sum_{s'} T_{ss'}^a [R(s', a, s) + V^\pi(s')] \right] \]

- to evaluate a policy, we need to solve the above equation, i.e. find the self-consistent state values

- options for policy evaluation
  - exhaustive tree search - outwards, inwards, depth-first
  - linear solution in 1 step
  - value iteration: iterative updates
  - experience sampling
Solving the Bellman Equation

Option 1: turn it into update equation

Option 2: linear solution  
(w/ absorbing states)

\[
V(s) = \sum_a \pi(a, s_t) \left[ \sum_{s'} T^\pi_{ss'} [R(s', a, s) + V(s')] \right]
\]

\[
\Rightarrow v = R^\pi + T^\pi v
\]

\[
\Rightarrow v^\pi = (I - T^\pi)^{-1} R^\pi \quad O(|S|^3)
\]
Solving the Bellman Equation

Option 1: turn it into update equation

\[ V^{k+1}(s) = \sum_{a} \pi(a, s_t) \left[ \sum_{s'} T_{ss'}^{a} \left[ R(s', a, s) + V^k(s') \right] \right] \]

Option 2: linear solution (w/ absorbing states)

\[ V(s) = \sum_{a} \pi(a, s_t) \left[ \sum_{s'} T_{ss'}^{a} \left[ R(s', a, s) + V(s') \right] \right] \]

\[ \Rightarrow \mathbf{v} = R^\pi + T^\pi \mathbf{v} \]

\[ \Rightarrow \mathbf{v}^\pi = (I - T^\pi)^{-1} R^\pi \quad O(|S|^3) \]
Policy update

Given the value function for a policy, say via linear solution

\[
V^\pi(s) = \sum_a \pi(a|s) \left[ \sum_{s'} T_{ss'}^a \left[ R(s',a,s) + V^\pi(s') \right] \right]
\]

Given the values V for the policy, we can improve the policy by always choosing the best action:

\[
\pi'(a|s) = \begin{cases} 
1 & \text{if } a = \arg\max_a Q^\pi(s,a) \\
0 & \text{else}
\end{cases}
\]

It is guaranteed to improve:

\[
Q^\pi(s, \pi'(s)) = \max_a Q^\pi(s,a) \geq Q^\pi(s, \pi(s)) = V^\pi(s)
\]

for deterministic policy

\[
\pi_0(a|s) = \begin{cases} 
1 & \text{if } a = \arg\max_a Q^\pi(s,a) \\
0 & \text{else}
\end{cases}
\]

\[
\pi_0(a|s) = \begin{cases} 
1 & \text{if } a = \arg\max_a Q^\pi(s,a) \\
0 & \text{else}
\end{cases}
\]
**Policy iteration**

Policy evaluation

\[ \mathbf{v}^\pi = (I - T^\pi)^{-1} \mathbf{R}^\pi \]

\[
\pi(a|s) = \begin{cases} 
1 & \text{if } a = \arg\max_a \sum_{s'} T_{ss'}^a \left[ R_{ss'}^a + V^{\pi}(s') \right] \\
0 & \text{else}
\end{cases}
\]
Policy iteration

\[ v^\pi = (I - T^\pi)^{-1} R^\pi \]

Policy evaluation

\[ \pi(a|s) = \begin{cases} 1 & \text{if } a = \text{argmax}_a \sum_{s'} T_{ss'}^a \left[ R_{ss}^a + V^{\pi}(s') \right] \\ 0 & \text{else} \end{cases} \]

greedy policy improvement
Policy iteration

Policy evaluation

\[ v^\pi = (I - T^\pi)^{-1} R^\pi \]

Value iteration

\[ V^*(s) = \max_a \sum_{s'} T_{ss'}^a [R_{ss}^a + V^*(s')] \]

greedy policy improvement

\[ \pi(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_a \sum_{s'} T_{ss'}^a [R_{ss}^a + V^{\pi}(s')] \\ 0 & \text{else} \end{cases} \]
Model-free solutions

- So far we have assumed knowledge of R and T
  - R and T are the ‘model’ of the world, so we assume full knowledge of the dynamics and rewards in the environment

- What if we don’t know them?

- We can still learn from state-action-reward samples
  - we can learn R and T from them, and use our estimates to solve as above
  - alternatively, we can directly estimate V or Q
Solving the Bellman Equation

Option 3: sampling

\[ V(s) = \sum_a \pi(a, s_t) \left[ \sum_{s'} T_{ss'}^a [R(s', a, s) + V(s')] \right] \]

So we can just draw some samples from the policy and the transitions and average over them:

\[ a = \sum_{k} f(x_k) p(x_k) \]

\[ x^{(i)} \sim p(x) \rightarrow \hat{a} = \frac{1}{N} \sum_i f(x^{(i)}) \]
Solving the Bellman Equation

Option 3: sampling

So we can just draw some samples from the policy and the transitions and average over them:

\[
\hat{a} = \frac{1}{N} \sum_{i} f(x^{(i)})
\]

\[
a = \sum_{k} f(x_k) p(x_k)
\]

\[
x^{(i)} \sim p(x) \rightarrow \hat{a} = \frac{1}{N} \sum_{i} f(x^{(i)})
\]
Solving the Bellman Equation

Option 3: sampling
	his is an expectation over policy and transition samples.

So we can just draw some samples from the policy and the transitions and average over them:

\[
\hat{a} = \frac{1}{N} \sum_{i} f(x^{(i)})
\]

\[
a = \sum_{k} f(x_k) p(x_k)
\]

\[
x^{(i)} \sim p(x) \rightarrow \hat{a} = \frac{1}{N} \sum_{i} f(x^{(i)})
\]
Solving the Bellman Equation

Option 3: sampling

this is an expectation over policy and transition samples.

So we can just draw some samples from the policy and the transitions and average over them:

\[ a = \sum_{k} f(x_k)p(x_k) \]

\[ x^{(i)} \sim p(x) \rightarrow \hat{a} = \frac{1}{N} \sum_{i} f(x^{(i)}) \]

more about this later...
Learning from samples

A new problem: exploration versus exploitation
Monte Carlo

- **First visit MC**
  - randomly start in all states, generate paths, average for starting state only

\[
\mathcal{V}(s) = \frac{1}{N} \sum_i \left\{ \sum_{t'=1}^T r_{t'} | s_0 = s \right\}
\]

- **More efficient use of samples**
  - Every visit MC
  - Bootstrap: TD
  - Dyna

- **Better samples**
  - on policy versus off policy
  - Stochastic search, UCT...
Update equation: towards TD

Bellman equation

\[ V(s) = \sum_a \pi(a, s) \left[ \sum_{s'} T_{ss'}^a \left[ R(s', a, s) + V(s') \right] \right] \]

Not yet converged, so it doesn’t hold:

\[ dV(s) = -V(s) + \sum_a \pi(a, s) \left[ \sum_{s'} T_{ss'}^a \left[ R(s', a, s) + V(s') \right] \right] \]

And then use this to update

\[ V^{i+1}(s) = V^i(s) + dV(s) \]
**TD learning**

\[
dV(s) = -V(s) + \sum_a \pi(a, s) \left[ \sum_{s'} T_{ss'}^a [R(s', a, s) + V(s')] \right]
\]
TD learning

\[ dV(s) = -V(s) + \sum_a \pi(a, s) \left[ \sum_{s'} T_{ss'}^a \left[ R(s', a, s) + V(s') \right] \right] \]

Sample

\begin{align*}
  a_t & \sim \pi(a|s_t) \\
  s_{t+1} & \sim T_{s_t,s_{t+1}}^{a_t} \\
  r_t & = R(s_{t+1}, a_t, s_t)
\end{align*}
TD learning

\[ dV(s) = -V(s) + \sum_a \pi(a, s) \left[ \sum_{s'} T_{ss'}^a \left( R(s', a, s) + V(s') \right) \right] \]

\[ a_t \sim \pi(a|s_t) \]

\[ s_{t+1} \sim T_{s_t,a_t}^{s_{t+1}} \]

\[ r_t = R(s_{t+1}, a_t, s_t) \]

\[ \delta_t = -V_{t-1}(s_t) + r_t + V_{t-1}(s_{t+1}) \]
### TD Learning

\[
dV(s) = -V(s) + \sum_a \pi(a, s) \left[ \sum_{s'} T_{ss'}^a [R(s', a, s) + V(s')] \right]
\]

- Sample

\[
\begin{align*}
  a_t & \sim \pi(a|s_t) \\
  s_{t+1} & \sim T_{s_t,a_t}^{s_{t+1}} \\
  r_t & = R(s_{t+1}, a_t, s_t)
\end{align*}
\]

\[
\delta_t = -V_{t-1}(s_t) + r_t + V_{t-1}(s_{t+1})
\]

\[
V^{i+1}(s) = V^i(s) + dV(s)
\]

\[
V_t(s_t) = V_{t-1}(s_t) + \alpha \delta_t
\]
**TD learning**

\[
\begin{align*}
\pi_t(a|s_t) & \sim \pi(a|s_t) \\
\mathcal{T}_{s_t,s_{t+1}}^{a_t} & \sim \mathcal{T}_{s_t,s_{t+1}}^{a_t} \\
R(s_{t+1}, a_t, s_t) &= R(s_{t+1}, a_t, s_t) \\
\delta_t &= -V_t(s_t) + r_t + V_t(s_{t+1}) \\
V_{t+1}(s_t) &= V_t(s_t) + \alpha \delta_t
\end{align*}
\]
Aside: what makes a TD error?

- unpredicted reward expectation change
- disappears with learning
- stays with probabilistic reinforcement
- sequentiality
  - TD error vs prediction error
- see Niv and Schoenbaum 2008

Schultz et al.
The effect of bootstrapping

Markov (every visit)
\[ V(B) = \frac{3}{4} \]
\[ V(A) = 0 \]

TD
\[ V(B) = \frac{3}{4} \]
\[ V(A) = \frac{3}{4} \]

- Average over various bootstrappings: \( TD(\lambda) \)

after Sutton and Barto 1998
Actor-critic

- policy and value separately parameterised

\[ \delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \]

\[ w(s, a) \leftarrow w(s, a) + \beta \delta_t \]

\[ w(s, a) \leftarrow w(s, a) + \beta \delta_t (1 - \pi(s, a)) \]

\[ \pi(a|s) = \frac{e^{w(s,a)}}{\sum_{a'} e^{w(s,a')}} \]
SARSA

- Do TD for state-action values instead:

\[ Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \]

\[ s_t, a_t, r_t, s_{t+1}, a_{t+1} \]

- convergence guarantees - will estimate \( Q^\pi(s, a) \)
Learn off-policy
• draw from some policy
• “only” require extensive sampling

\[
Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]
\]

update towards optimum

will estimate \( Q^*(s, a) \)
Learning in the wrong state space

- states = distance from goal
- state-space choice crucial
  - too big -> curse of dimensionality
  - too small -> can’t express good policies
  - unsolved problem
- humans in tasks have to infer state-space
Neural network approximations

- So far: look-up tables

- Parametric value functions

- Humans and animals generalize
Hierarchical decompositions

- **Subtasks stay the same**
  - Learn subtasks
  - Learn how to use subtasks

- **Macroactions**
  - ‘go to door’
  - search goal

- **Humans establish subgoals on-line**
  - how is not yet known
Learning a model

- So far we’ve concentrated on model-free learning
- What if we want to build some model of the environment?

\[
V(s) = \sum_a \pi(a, s) \left( \sum_{s'} T_{ss'}^a \left[ R(s', a, s) + V(s') \right] \right)
\]

- Count transitions

\[
\hat{T}_{ss'}^a = \frac{\sum_t 1(s_t = s, a_t = a, s_{t+1} = s')}{\sum_t 1(s_t = s, a_t = a)}
\]

- Average rewards

\[
\hat{R}_{ss'}^a = \frac{\sum_t r_t 1(s_t = s, a_t = a, s_{t+1} = s')}{\sum_t 1(s_t = s, a_t = a, s_{t+1} = s')}
\]
Dyna

Combine model estimation with TD learning

\[ V_{t+1}(s_t) = V_t(s_t) + \alpha \delta_t \]

Generate extra experience samples from estimated model

Sutton & Barto 1998, Figure 9.5
Conclusion I

- Long-term rewards have internal consistency
- This can be exploited for solution
- Exploration and exploitation trade off when sampling
- Clever use of samples can produce fast learning
  - Brain most likely does something like this