Reinforcement Learning
I: Theory

Quentin Huys

Wellcome Trust Centre for Neuroimaging, UCL
Gatsby Computational Neuroscience Unit, UCL
Guy’s and St. Thomas’ Hospital NHS Trust

Psychiatrische Universitätsklinik, University of Zurich
Theoretical Neuromodeling Unit, ETH

Advanced Course in Computational Neuroscience, Bedlewo, Poland, August 2012
Overview

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

- Some behavioural considerations
  - a few behavioural and neurobiological examples & applications
  - psychopathology

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

- Supervised
- Unsupervised
- Reinforcement learning
Setup

After Sutton and Barto 1998

\[
\{a_t\} \leftarrow \text{argmax} \sum_{t=1}^{\infty} r_t
\]
State space
A Markov Decision Problem

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

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

Action left

Action right

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

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

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

Noisy: plants, environments, agent
Actions

Action left

Action right

Noisy: plants, environments, agent

Absorbing state -> max eigenvalue < 1
Markov state-space descriptions

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

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

Velocity
Markov state-space descriptions

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

Velocity

\[ s' = [\text{position}] \rightarrow s' = \begin{bmatrix} \text{position} \\ \text{velocity} \end{bmatrix} \]
A Markov Decision Process (MDP) is defined by:

- \( s_t \in \mathcal{S} \) (state at time \( t \))
- \( a_t \in \mathcal{A} \) (action at time \( t \))
- \( \mathcal{T}_{ss'}^a = p(s_{t+1} | s_t, a_t) \) (transition probability)
- \( r_t \sim \mathcal{R}(s_{t+1}, a_t, s_t) \) (reward probability)
- \( \pi(a | s) = p(a | s) \) (policy function)
$$s_t \in S$$
$$a_t \in A$$
$$\mathcal{T}^a_{ss'} = p(s_{t+1} | s_t, a_t)$$
$$r_t \sim \mathcal{R}(s_{t+1}, a_t, s_t)$$
$$\pi(a | s) = p(a | s)$$
MDP

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

\[ w^d \]
Decision tree

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

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

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

8
64
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
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

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

- 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
    \[ T \sim \frac{1}{\tau} e^{t/\tau} \]
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

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

\[ t \]

\[ T \]

\[ e^{t/\tau} \]
Markov Decision Problems

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

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

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

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

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

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

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

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

\[
= \sum_{a_t} \mathcal{T}_{s_{t+1} s_{t+2}} \mathcal{R}(s_{t+1}, a_t, s_t)
\]
Bellman equation

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

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

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

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

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

All future reward from state \( s \) = \( E \) Immediate reward + All future reward from next state \( s' \)
Bellman Equation

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

All future reward from state $s$ = E Immediate reward + All future reward from next state $s'$
\[ V^\pi(s) = \sum_a \pi(a|s) \left[ \sum_{s'} T_{ss'}^a \left[ R(s', a, s) + V^\pi(s') \right] \right] \]

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

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

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

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

Option 1: turn it into update equation

\[
V^{k+1}(s) = \sum_a \pi(a, s_t) \left[ \sum_{s'} T^a_{ss'} \left[ R(s', a, s) + V^k(s') \right] \right]
\]

Option 2: linear solution (w/ absorbing states)

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

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

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

\(O(|S|^3)\)
Policy update

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

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

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

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

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
Policy iteration

Policy evaluation

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

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

Policy evaluation

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

greedy policy improvement

\[
\pi(a|s) = \begin{cases} 
1 & \text{if } a = \arg\max_a \sum_{s'} T^a_{ss'} [R^a_{ss'} + V^{p_i}(s')] \\
0 & \text{else}
\end{cases}
\]
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

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

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

\[ a = \sum_k f(x_k)p(x_k) \]

\[ x^{(i)} \sim p(x) \rightarrow \hat{a} = \frac{1}{N} \sum_i f(x^{(i)}) \]
Option 3: sampling

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

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:

\[
a = \sum_k f(x_k) p(x_k)
\]

\[
x^{(i)} \sim p(x) \rightarrow \hat{a} = \frac{1}{N} \sum_i f(x^{(i)})
\]
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
  \[
  V(s) = \frac{1}{N} \sum_i \left\{ \sum_{t'=1}^T r_{t'}^i | s_0 = s \right\}
  \]

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

- **Better samples**
  - on policy versus off policy
  - UCB, UCT, BOSS...
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

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

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

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

Sample

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

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

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

Sample

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

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

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

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

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

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

\begin{align*}
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) \\
\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) = 3/4
V(A) = 0

TD
V(B) = 3/4
V(A) = ~3/4

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

after Sutton and Barto 1998
Actor-critic

- policy and value separately parametrised

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

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

\[ \pi(a|s) = \frac{e^{w(s,a)}}{\sum_{a'} e^{w(s,a')}} \]
- 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 \(Q^\pi(s, a)\)
Q learning: off-policy

- Learn off-policy
  - draw from some policy
  - “only” require extensive sampling

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

- 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

\[ Q(s, a; \theta) \]
Hierarchical decompositions

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

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

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

- Average rewards

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