Decision Making III

Katie DC
Markov Models

Markov Decision Processes (MDP) | Partially Observable MDP | Reinforcement Learning

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
<th>Reward</th>
<th>Transition Probability</th>
<th>State</th>
<th>Action</th>
<th>Reward</th>
<th>Transition Probability</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>0.9</td>
<td>0.1</td>
<td>2</td>
<td>B</td>
<td>0.4</td>
<td>0.6</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>0.6</td>
<td>0.4</td>
<td>3</td>
<td>B</td>
<td>0.3</td>
<td>0.7</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>0.1</td>
<td>0.9</td>
<td>1</td>
<td>B</td>
<td>0.3</td>
<td>0.7</td>
<td>2</td>
</tr>
</tbody>
</table>

Uncertainty in effects of actions | Uncertainty in current state | Uncertainty in model
Challenges for Reinforcement Learning

1. **Exploration** of the world must be balanced with **exploitation** of the knowledge gained through previous experience.

2. Reward may be received long after important choices have been made, so credit must be assigned to earlier decisions.

3. Must **generalize** from limited experience.

There are many solutions to this problem!

For a comprehensive overview, check out *Reinforcement Learning: An Introduction* by Sutton and Barto.
Reinforcement Learning

Goal: determine optimal actions that maximize expected reward

\[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \]

Solution methods:
- Model based
- Model free

not given T or R directly
must learn through experience
Q-learning: Model-free method: Q-Learning

• Agent gathers experience: \((s, a, r, s')\)
• Q-function returns the expected reward of that action at that state
• Temporal Differences to estimate optimal value \(Q^*\) for each state
• Agent maintains Q-table of all \(Q\) values for each state \(s\) and action \(a\)
suppose we have a random variable $X$; how to estimate mean given samples $x_1:n$?

$$\hat{x}_n = \frac{1}{n} \sum_{i=1}^{n} x_i$$

we can show that

$$\hat{x}_n = \hat{x}_{n-1} + \frac{1}{n} (x_n - \hat{x}_{n-1})$$

$\alpha(n) \rightarrow$ learning rate, often constant

$$\hat{x} \leftarrow \hat{x} + \alpha (x - \hat{x})$$

$\triangleleft$ temporal difference error
Incremental Estimation Example

Current mean estimate $\hat{m} = 3$

New sample: $x = 7$

$\hat{m} \leftarrow \hat{m} + \alpha (x - \hat{m})$

if $\alpha = 0.1$

$\hat{m} \leftarrow 3 + 0.1 (7 - 3) = 3.4$

if $\alpha = 0.5$

$\hat{m} \leftarrow 3 + 0.5 (7 - 3) = 5$
Q-Learning (1)

Key idea: apply incremental estimation to Bellman eq.

\[ Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s'|s,a) V(s') \]

\[ = R(s,a) + \gamma \sum_{s'} T(s'|s,a) \max_{a'} Q(s',a') \]

since we don't have \( T \) or \( R \), use observed next states \( s' \) and \( r \) to estimate the \( Q \) values

Use the following incremental update rule:

\[ Q(s,a) \leftarrow Q(s,a) + \alpha (r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \]
Q-Learning (2)
Q-Learning Algorithm

```
function Qlearning
  t ← 0
  s₀ ← initial state
  Initialize Q
loop
  Choose action aₜ based on Q and some exploration strategy
  Observe new state sₜ₊₁ and reward rₜ
  Q(sₜ, aₜ) ← Q(sₜ, aₜ) + α \left( rₜ + γ \max_a Q(sₜ₊₁, a) - Q(sₜ, aₜ) \right)
  t ← t + 1
```
Q-Learning Challenges

• How should an agent decide which actions to choose to explore?
• One way to define probabilistic exploration strategy, using the Boltzmann distribution:

\[ P(a|s) = \frac{e^{Q(s,a)/k}}{\sum_j e^{Q(s,a_j)/k}} \]

The \( k \) parameter (called temperature) controls probability of picking non-optimal actions. If \( k \) is large, all actions are chosen uniformly (explore), if \( k \) is small, then the best actions are chosen.
Q-Learning Challenges

• How should an agent decide which actions to choose to explore?
• The Q Table can be thought of as a cheat sheet. How many states and actions must be stored for a game of chess?
• Another issue generally in RL: How to know if reward is correct? How do we best shape the reward to get a desirable outcome? Is that okay?
DQN: approximate Q with deep network

• target = \( R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \)

• \( Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha \text{[target]} \)

• Goal is to approximate our Q table with a deep network that will act as a Q Function

```
function DQN
s_0 ← initial state
Initialize Q_0
for k = 1, 2, ...
    Choose action \( a_t \) / Observe new state \( s_{t+1} \) and reward \( r_t \)
    target = \( R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \)
    \( \theta_{k+1} \leftarrow \theta_k + \alpha \nabla E_{s' \sim P(s'|s, a)} [Q_{\theta}(s, a) - \text{target}(s')] \big|_{\theta=\theta_k} \)
    \( s \leftarrow s' \)
```
DQN Challenges

• Deep learning works for supervised learning under these conditions:
  • Samples are i.i.d., meaning that each batch has the same distribution and all samples are independent within the batch
  • For some input, the label is consistent across time

• In RL, these typically do not hold
  • Target is unstable!
  • Not iid: when parameters are updated, local states are also effected
  • Actions are chosen by estimated Q (we choose what to explore or exploit), this means our target output (action) is constantly changing as well

```python
function DQN
  s_0 \leftarrow \text{initial state}
  \text{Initialize } Q_0
  \text{for } k = 1, 2, ...
  \text{Choose action } a_t / \text{Observe new state } s_{t+1} \text{ and reward } r_t
  \text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')
  \theta_{k+1} \leftarrow \theta_k + \alpha \nabla E_{s' \sim P(s'|s,a)} [Q_\theta(s, a) - \text{target}(s')]_{\theta=0}
  s \leftarrow s'```

non-stationary
correlation with trajectories
• Experience Replay
  • Say you store $10^6$ transitions and use a batch size of 32 to train the network.
  • Sampling from this buffer forms a dataset that is close to iid and therefore stable

• Target network:
  • Use two deep networks! $\theta^-$ and $\theta$.
  • First retrieves Q values and the second updates in the training. By temporarily fixing the Q-value targets, the moving target issue is solved.

• $L_i(\theta_i) = \mathbb{E}_{s,a,s',r \sim D} \left( r + \gamma \max_{a'} Q(s', a'; \theta^-_i) - Q(s, a; \theta_i) \right)^2$
Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
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 $\epsilon$ select a random action $a_t$
otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$
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 $(\phi_i, a_j, r_j, \phi_{j+1})$ from $D$

Set $y_j = \begin{cases} \ \ \ \ \ \ r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters $\theta$
Every $C$ steps reset $\hat{Q} = Q$

End For
End For

$P(a|s) = \frac{\epsilon}{m} + 1 - \epsilon$, if $a^*,$ otherwise