Reinforcement Learning and Beyond:
I. Introduction to Reinforcement Learning

Francisco S. Melo

INESC-ID, Portugal
Outline

1 Dynamic Programming

2 Temporal Difference Learning

3 “Batch” RL

4 The actor-critic architecture

5 References
Back to the RL Setting
Optimality

- Reward function “encodes” the task of the agent
- The agent must choose the actions that maximize the total (discounted) reward,
  \[ \sum_{t=0}^{\infty} \gamma^t r_t \]

- Policies map states to actions
- The value of a policy \( \pi \) at state \( x \) is
  \[ V^\pi(x) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_t \mid X_0 = x \right] \]
Optimality (cont.)

- We want the **optimal policy**, $\pi^*$, such that

  $$V^{\pi^*}(x) \geq V^\pi(x),$$

  for all $x$

- In other words, we want the policy whose **value** at every state is maximal

**THE RL QUESTION:** How to compute $\pi^*$ when we do not know $P$ or $r$?
But First...

If we do know $P$ and $r$...

- How can we compute the optimal policy, $\pi^*$? (policy optimization)

... or even...

- If we are given a policy $\pi$, how good is it? (policy evaluation)

Both problems can be solved using dynamic programming.
The Bellman Equation

Bellman Optimality Principle

If you know you will behave optimally after some time $T$, you need only to reason about how to behave optimally from now to $T$.

- The function $V^{\pi^*}$ verifies the Bellman optimality equation:

$$ V^{\pi^*}(x) = \max_{a \in A} \mathbb{E} \left[ r(x, a) + \gamma V^{\pi^*}(y) \right] $$

- We can also define a function $Q^*$ as

$$ Q^*(x, a) = \mathbb{E} \left[ r(x, a) + \gamma V^{\pi^*}(y) \right] $$
Useful Relations

Some useful relations:

- Relation between $V^{\pi^*}$ and $Q^*$:
  \[ V^{\pi^*}(x) = \max_{a \in A} Q^*(x, a) \]

- Relation between $V^{\pi^*}$ and $\pi^*$
  \[ \pi^*(x) = \arg\max_{a \in A} \mathbb{E} \left[ r(x, a) + \gamma V^{\pi^*}(y) \right] \]

- Relation between $Q^*$ and $\pi^*$
  \[ \pi^*(x) = \arg\max_{a \in A} Q^*(x, a) \]
Solving Policy Evaluation/Optimization

If we do know $P$ and $r$...

- Q: How can we compute the optimal policy, $\pi^*$? (policy optimization)
- A: We compute $Q^*$ and use the relation

$$\pi^*(x) = \arg\max_{a \in A} Q^*(x, a)$$

- Q: If we are given a policy $\pi$, how good is it? (policy evaluation)
- A: We compute the corresponding value-function, $V^\pi$
The Dynamic Programming Approach

Both $Q^*$ and $V^\pi$ are fixed-points:

\[
Q^*(x, a) = \mathbb{E} \left[ r(x, a) + \gamma \max_{b \in A} Q^*(y, b) \right]
\]

\[
V^\pi(x) = \mathbb{E}_\pi \left[ r(x, a) + \gamma V^\pi(y) \right]
\]

Dynamic programming uses these relations to estimate $Q^*$ and $V^\pi$:

\[
Q^{(k+1)}(x, a) = \mathbb{E} \left[ r(x, a) + \gamma \max_{b \in A} Q^{(k)}(y, b) \right]
\]

\[
V^{(k+1)}(x) = \mathbb{E}_\pi \left[ r(x, a) + \gamma V^{(k)}(y) \right]
\]
Outline

1. Dynamic Programming
2. Temporal Difference Learning
3. “Batch” RL
4. The actor-critic architecture
5. References
Model-Based Learning

- Given an MDP \((\mathcal{X}, \mathcal{A}, P, r, \gamma)\), if \(P\) and \(r\) are known, we can apply DP.
- If \(P\) and/or \(r\) are not known, we can estimate these parameters from experience.
The need for exploration

The first encounter with the Exploration vs. Exploitation tradeoff...

- In learning P, the agent must visit each state-action pair infinitely often
- This means that the agent must ensure sufficient exploration
- Insufficient exploration may prevent convergence of \( \hat{p} \)

This would be even more critical if the policy used to sample the process were to improve over time...
Temporal Difference Learning

- If $z^* = \mathbb{E}_x [h(x)]$ then, given $t$ samples $\{h(x_i), i = 1, \ldots, t\}$,
  \[ z^{(t)} = \frac{1}{t} \sum_i h(x_i) \]

- Iteratively,
  \[ z^{(t+1)} = z^{(t)} + \frac{1}{t+1} (h(x_{t+1}) - z^{(t)}) \]

- In the DP iterations, we compute expectations, so a similar approach should be possible.
Temporal Difference Learning (cont.)

- To compute $V^\pi$, given sample sequences $\{x_t\}$ and $\{r_t\}$, this yields
  \[
  V^{(t+1)}(x_t) = V^{(t)}(x_t) + \alpha_t \left[ r_t + \gamma V^{(t)}(x_{t+1}) - V^{(t)}(x_t) \right]
  \]

- Similarly, to compute $Q^*$, given sequences $\{x_t\}$, $\{a_t\}$ and $\{r_t\}$,
  \[
  Q^{(t+1)}(x_t, a_t) = Q^{(t)}(x_t, a_t) + \alpha_t \left[ r_t + \gamma \max_{b \in A} Q^{(t)}(x_{t+1}, b) - Q^{(t)}(x_t, a_t) \right]
  \]

- The algorithm to compute $V^\pi$ is called TD(0)
- The algorithm to compute $Q^*$ is called Q-learning
- The quantities $\alpha_t$ are step-sizes
- The quantities multiplying $\alpha_t$ are called temporal differences
MDPs with Large/Infinite State-Space

- Both TD(0) and Q-learning update estimates $V^{(t)}$ and $Q^{(t)}$ component-wise.
- If $\mathcal{X}$ is large/infinite, explicit representation of $V^{(t)}$ and $Q^{(t)}$ is computationally too expensive (or even impossible).
- Some form of compact representation is necessary.

### Linear Function Approximation

We adopt parameterized representations $V_\theta$ and $Q_\theta$ for $V^\pi$ and $Q^*$, respectively. Concretely, given a family of real-valued functions $\{\phi_i, i = 1, \ldots, M\}$,

\[
V_\theta(x) = \phi^\top(x)\theta \\
Q_\theta(x, a) = \phi^\top(x)\theta_a
\]
Updates are now performed upon the parameter $\theta$

Components of $\theta$ updated proportionally to the corresponding component of $\phi(x)$

The modified TD(0) update

$$
\theta(t+1) = \theta(t) + \alpha_t \phi(x_t) \left[ r_t + \gamma v_{\theta(t)}(x_{t+1}) - v_{\theta(t)}(x_t) \right]
$$

The modified Q-learning update

$$
\theta_{at}^{(t+1)} = \theta_{at}^{(t)} + \alpha_t \phi(x_t) \left[ r_t + \gamma \max_{b \in A} q_{\theta(t)}(x_{t+1}, b) - q_{\theta(t)}(x_t, a_t) \right]
$$
The fixed-point of the algorithm is such that the DP iteration is orthogonal to $\mathcal{V}$. 

**Legend:**
- Algorithm iter.
- DP iter.
- Projection
Outline

1. Dynamic Programming
2. Temporal Difference Learning
3. “Batch” RL
4. The actor-critic architecture
5. References
The idea behind Monte-Carlo RL

Each sample is used as soon as it becomes available

Only after a “batch” of samples is collected does the agent use the data

THE RL BROTHERS FACTORY

TD

MC

AAMAS 2010 (Toronto)  Tutorial AT4: RL & Beyond  11.05.10  20 / 36
Monte-Carlo vs. TD

- In TD-learning, the agent updates the estimate of the target function after each sample.
- In Monte-Carlo, the agent collects the data from a complete \textit{episode} before updating.
- Monte-Carlo methods process a whole “batch” of data in each update.
- In Monte-Carlo, the data from each episode works as a \textit{sample run}. 
Evaluating a policy

- Let $\mathcal{M} = (\mathcal{X}, \mathcal{A}, P, r, \gamma)$ be an MDP and $\pi$ a policy to be evaluated.
- Let $S = \{x_0, a_0, r_0, x_1, \ldots, x_N\}$ be a $N$-step sample sequence (episode).
- For $x \in S$, the return at $x$ is

$$R(x) = \sum_{t=\tau_x}^{N} \gamma^{t-\tau_x} r_t,$$

where $\tau_x$ is the first-time state $x$ is visited in $S$.
- $R(x)$ is a sample of $V^\pi(x)$.  

After each episode, the agent updates the value of every state appearing in the episode as

\[ V(x) \leftarrow V(x) + \frac{1}{k(x) + 1} (R(x) - V(x)) \]

\[ k(x) \leftarrow k(x) + 1 \]

- \( k(x) \) is the number of updates to \( V(x) \)
- The estimate \( V(x) \) is just the sample mean of the observed returns at \( x \)
To compute $Q^*$, we need episodes generated with $\pi^*$ (unknown)

We can perform one step of policy optimization between every two episodes.

The algorithm thus obtained can be seen as “MC policy iteration”

The policy optimization steps must still ensure enough exploration.

Exploration can be implemented with exploring starts (the first action is chosen randomly).
Leveraging regression methods to RL

- MC approach is straightforward and simple
- Closely related to TD-learning (see Sutton and Barto, 1998)
- However, as before, $V$ and $Q$ cannot be represented explicitly in problems with infinite $\mathcal{X}$
- The use of batches of data in each update suggests formulation of RL as a regression problem
- Powerful methods from general machine learning to be used in RL
Leveraging regression methods to RL (cont.)

- Recall the fixed point equations for $V^\pi$ and $Q^*$:

  \[
  V^\pi(x) = \mathbb{E}_\pi [r(x, a, y) + \gamma V^\pi(y)]
  \]

  \[
  Q^*(x, a) = \mathbb{E} \left[ r(x, a, y) + \gamma \max_{b \in \mathcal{A}} Q^*(y, b) \right]
  \]

- We can rewrite these equations as

  \[
  0 = \mathbb{E}_\pi [r(x, a, y) + \gamma V^\pi(y) - V^\pi(x)]
  \]

  \[
  0 = \mathbb{E} \left[ r(x, a, y) + \gamma \max_{b \in \mathcal{A}} Q^*(y, b) - Q^*(x, a) \right]
  \]
Defining the error term

From the previous equations...

- Given an estimate $\nu$ of $V^\pi$, we define the error term

$$E(\nu) = \|E_\pi [r(x, a, y) + \gamma \nu(y) - \nu(x)]\|_p$$

- Given an estimate $q$ of $Q^*$, we define the error term

$$E(q) = \|E \left[ r(x, a, y) + \gamma \max_{b \in A} q(y, b) - q(x, a) \right]\|_p$$

- In both expressions, $\| \cdot \|_p$ denotes some adequate $p$-norm
Fitted VI
Regression for policy evaluation

- Let $\mathcal{D} = \{(x_0, a_0, r_0, y_0), \ldots, (x_N, a_N, r_N, y_N)\}$ be a set of transition triplets sampled from the MDP using the desired policy $\pi$
- Let $v^{(0)}$ be some initial estimate for $V^\pi$
- At iteration $k$ of the algorithm, and for every sample transition $(x_i, a_i, r_i, y_i)$, let
  $$\vartheta_i = r_i + \gamma v^{(k-1)}(y_i)$$
- Compute $v^{(k)}$ to minimize the sample error
  $$\hat{E}(v) = \frac{1}{N} \sum_{i=0}^{N} |\vartheta_i - v(x_i)|^p$$
Fitted VI (cont.)

- The algorithm thus obtained is called fitted value iteration.
- Each iterate of the algorithm applies value iteration at the sampled points.
- It then uses the regression algorithm to generalize this update to the whole state-space.
- As in MC methods, fitted VI uses all the data in performing a single update.
- Being a regression algorithm, it does not exhibit the divergence problems observed in TD-learning.
Fitted QI
Regression for policy optimization

- Let $D = \{(x_0, a_0, r_0, y_0), \ldots, (x_N, a_N, r_N, y_N)\}$ be a set of transition triplets sampled from the MDP using some policy $\pi$.
- Let $q^{(0)}$ be some initial estimate for $Q^*$.
- At iteration $k$ of the algorithm, and for every sample transition $(x_i, a_i, r_i, y_i)$, let
  \[ \xi_i = r_i + \gamma \max_{a \in A} q^{(k-1)}(y_i, a) \]
- Compute $q^{(k)}$ so as to minimize the sample error
  \[ \hat{E}(q) = \frac{1}{N} \sum_{i=0}^{N} |\xi_i - q(x_i, a_i)|^p \]
If you really want to know...
A discussion on error bounds and error minimization

- The error $\mathcal{E}(q)$ measures how well a given function adheres to the Bellman optimality principle.

- It is known as the Bellman error or Bellman residual.

- Some works have implemented gradient methods to minimize the Bellman residual (Baird, 1995).

- Recent works (Munos and Szepesvári, 2008; Antos et al., 2006, 2008) have provided detailed analysis of Fitted-VI and Fitted-QI.
Outline

1. Dynamic Programming
2. Temporal Difference Learning
3. “Batch” RL
4. The actor-critic architecture
5. References
The actor-critic architecture

Bringing everything together...

\[ \text{ENVIRONMENT} \]

\[ (x, a, r, y) \]

\[ \text{CRITIC} \]

\[ \text{TD}(\lambda) \]

\[ \text{New policy} \]

\[ \pi_{\theta} \]

\[ \text{Policy evaluation} \]

\[ Q_{w^*} \]

\[ \theta_{t+1} = \theta_t + \alpha_tw^* \]
Outline

1. Dynamic Programming
2. Temporal Difference Learning
3. “Batch” RL
4. The actor-critic architecture
5. References
References


