Reinforcement Learning

Keith L. Downing

The Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
keithd@idi.ntnu.no

October 15, 2018
Focus on systems that seek to maximize or minimize some factor through a series of actions, not just maintain equilibrium.

Occasional feedback from an instructor, indicating only right/wrong, but not the correct answer/response for each context.

Problem solutions consist of many steps/stages, but the reinforcement (= reward or penalty) comes only intermittently, and maybe only at the end of a problem-solving sequence.

Credit assignment problem - how to give credit/blame to intermediate steps?

Well-documented theory (Reinforcement Learning, Sutton and Barto (1998, 2018)).

Typical usage: problems with a) limited possibilities for intermediate feedback, but b) concrete feedback at sequence end.

For example: games (Backgammon, Othello, Checkers...) and robotics.

RL often uses neural nets as the value function or policy.
Credit Assignment

Which states (actions) get credit?
How much?
Same amount for each on the path?

Win !!!

Keith L. Downing  Reinforcement Learning
Key Components of an RL System

1. **Policy** - mapping from states to (appropriate) actions
2. **Value Function** - mapping from states to their evaluations (a.k.a. values)
3. **Reward signal** - normally given intermittently.
4. **Model** of the Environment (optional) - probability distribution over rewards and next states given current state and action: $p(s', r | s, a)$
A complete state-value function covers all (of the many) states.

A good policy bases action choices on a good state-value function: choose the action that leads to the highest-valued, successor state.
Updating the Value Function via Experience

Reach Goal + Backup Reward

Reach Deadend + Backup Penalty

After Many Exploratory Rounds
Policies

- The RL system uses a **behavior policy** (i.e. a search strategy) for choosing actions in states (for most effective learning)
- while simultaneously building a **target policy** for choosing actions in states (for optimal problem-solving).
- **On-policy** methods $\rightarrow$ behavior policy $=$ target policy;
  **Off-policy** methods $\rightarrow$ behavior policy $\neq$ target policy

Value Functions

- Policies use a **state-value function** $=$ mapping: state $\rightarrow$ G (the total, expected **future** reinforcement for an agent currently in that state.)
- Alternatively, they may use an **action-value function** $=$ mapping: (state, action) $\rightarrow$ G.
Exploration -vs- Exploitation in AI

Common distinction in search-based problem solving = most AI.

**Explore**
- Assume that past experience has not achieved optimal results, so keep trying new things.
- "things" = actions from particular search states.

**Exploit**
- Put faith in past experience by repeating previous state-action pairs.
- This typically assumes that the strategy learned from past experience is now optimal, so there is no need to try new things.

- Normally, you want to keep exploring until you think you have an optimal strategy (e.g. target policy in RL).
- But if you are trying to optimize something (e.g. rewards) over the entire learning period, then the explore-exploit tradeoff becomes complex (see k-armed bandits).
Multi-Armed Bandit Problems

- A slot machine with K arms.
- Each move = pay 1 unit, pull one lever/arm.
- Goals:
  - Maximize your profit over N trials. Use a behavior policy to try different arms.
  - Learn which arm (A+) has best payoff. Target policy = pull A+ every time.
- Given this target policy, future visits to the casino can be very successful.

- Behavior policy = strategy for learning a good strategy (i.e., target policy)
- Other examples: Education policies, Drug trials on real patients,...
The Reinforcement in RL

- $r_t =$ reinforcement (a.k.a. reward) at time $t$.
- $G_t =$ cumulative reward from time $t$ until episode termination time ($T$). A.k.a. the return.

\[
G_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_T
\]

Normally in RL, future rewards are discounted by a factor, $\gamma$, yielding:

\[
G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T = G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}
\]

State-value function = mapping: states $\Rightarrow$ expected $G_t$, where actual moves taken and rewards achieved depend upon the policy, $\Pi$:

\[
V_\Pi(s) = E_\Pi\{G_t|s_t = s\} = E_\Pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k}|s_t = s\}
\]

Action-value function = mapping: (state, action) $\Rightarrow$ expected $G_t$, assuming $\Pi$.

\[
Q_\Pi(s, a) = E_\Pi\{G_t|s_t = s, a_t = a\} = E_\Pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k}|s_t = s, a_t = a\}
\]
Main Types of RL

1. Dynamic Programming (DP) - model based, bootstrapping
2. Monte Carlo Methods (MC) - experience-based, no bootstrapping, updates at episode end
3. Temporal Differencing (TD) - experience-based, bootstrapping, updates after each action

- Model-based - all probabilities of transitions between states given actions (and the ensuing rewards) are known ahead of time.
- Experience-based - those probabilities are learned by acting in the environment.
- Bootstrapping - values of one state (or action) are updated based on values of successor states or actions.
- Episode - a problem-solving instance, from start state to end state.
Basic Model-Free RL Architecture

Model-Free \( := \) Learning based on World Experience ("Sampling")

Keith L. Downing  Reinforcement Learning
RL in Search Space

- Non-final state
- Final state
- States with updated values
- Model

Keith L. Downing  Reinforcement Learning
Dynamic Programming (DP) Search Space

Update Step 0

Update Step 1

Update Step 2

Update Step 3
Monte Carlo (MC) Search Space

Sample Environment
One Episode
Update Values

Value update based on rewards after visit (with discounts)
Temporal Differencing (TD) Search Space

Sample Environment
K steps

Update
1 state
value

Bootstrapping
Value Updates

Update
next state
value after
next move

K = 3
Backup Schemes: DP, MC, TD

Dynamic Programming

Temporal Difference Methods
TD(1 step)  TD(2 step)  TD(n step)

Monte Carlo Methods
3 Successive Backup Rounds for 3 RL Approaches

Goal

Start

Dynamic Programming

Monte Carlo

Temporal Difference (2 step)

Episode #1

Episode #2

Episode #3

Keith L. Downing
Reinforcement Learning
Dynamic Programming (DP)

Theoretical basis for much of RL, but often impractical for hard problems.

The Bellman Equation

A set of constraints, one per state, s, that should hold, given policy Π:

\[ V_\Pi(s) = \sum_a \Pi(s, a) \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V_\Pi(s') \right] \]

where:

1. \( \Pi(s, a) = \) policy = prob choosing action a in state s.
2. \( \gamma = \) the discount factor.
3. The model
   - \( P_{ss'}^a = \) prob transitioning from state s to s’ on action a,
   - \( R_{ss'}^a = \) immediate reward received on that transition,

DP uses this as the basis for value updates.
**Bellman Example**

*Assumes \( \gamma = 1 \) (no discounting)*

\[
V(s) = \begin{cases} 
0.3 \times [0.6 \times (2 + V(w)) + 0.4 \times (3 + V(x))] \\
+ 0.7 \times [0.9 \times (4 + V(y)) + 0.1 \times (5 + V(z))] 
\end{cases}
\]
Bellman Equation for Action Values, $Q(s, a)$

$Q_{\Pi}(s, a) =$ expected return when choosing action $a$ in state $s$ while following policy $\Pi$.

$$Q_{\Pi}(s, a) = \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma \sum_{a'} \Pi(s', a') Q_{\Pi}(s', a')]$$

where:

1. $\Pi(s', a') =$ policy = prob choosing action $a'$ in state $s'$.
2. $\gamma =$ the discount factor.
3. The model
   1. $P_{ss'}^a =$ prob transitioning from state $s$ to $s'$ on action $a$,
   2. $R_{ss'}^a =$ immediate reward received on that transition,
For a state-value function, $V(s)$

$$V_*(s) = \max_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V_*(s')]$$

Under an optimal policy, the value of a state equals the best of the expected returns among all possible actions from that state.

For an action-value function, $Q(s,a)$

$$Q_*(s, a) = \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma \max_{a'} Q_*(s', a')]$$

(1)

Under an optimal policy, the value of an action ($a$) equals the expected value (averaged over all next states) of the best expected returns from each of those states.

- Any policy that is greedy w.r.t. $V_*(s)$ or $Q_*(s, a)$ is optimal.
- Greedy $\rightarrow$ pick action $a^*$ in $s$ with max $Q_*$ value, or $a^*$ that leads to next state with max $V_*$ value.
Satisfying the Bellman Equations

- In DP, the combination of perfect model and manageable number of states makes it possible to satisfy the Bellman equations using either linear programming techniques or other iterative updates. Eventually, the state values converge to a set that satisfies all Bellman equalities.
- But many real-life problems have too many states and/or imperfect models (if they have models at all).
- These problems, when solved with RL methods, can still utilize the basic structure of the Bellman equations as the basis for value updates.
- Instead of a model, other RL methods (MC, TD) explore the world, i.e. attempt to solve the problem many times.
- During exploration, they can gather statistics that can approximate the underlying model.
Generalized Policy Iteration (GPI): 2 alternating calcs

Policy Evaluation (PE)

Updating $V(s)$ based on the current policy, $\Pi$, using, e.g., the Bellman optimality equation.

$$V(s) \leftarrow \sum_{s'} P_{ss'}^{\Pi(s)} \left[ R_{ss'}^{\Pi(s)} + \gamma V(s') \right]$$

Policy Improvement (PI)

Updating of $\Pi$ based on $V(s)$; e.g., greedy selection of the action leading to the highest-valued successor.

$$\Pi(s) \leftarrow \text{argmax}_a P_{ss'}^a \left[ R_{ss'}^a + \gamma V(s') \right]$$

- In contrast to Actor-Critic model (ACM, shown later), the PI step directly involves $V(s)$.
- In ACM, actor maintains $\Pi$ and critic handles $V(s)$, but the actor only sees an error term, not $V(s)$. 

Keith L. Downing
Reinforcement Learning
Generalized Policy Iteration

Policy Evaluation = Value-Function Updating

Policy Improvement
Many central RL concepts and methods – $v(s)$, $q(s,a)$, $\Pi$, $G_t$, GPI – stem from DP.

But DP is impractical for most real problems, due to:
- huge state spaces
- lack of models

The more practical RL variants, MC and TD, explore within state spaces, but:
- cannot expect to ever visit ALL states, and
- base GPI on incomplete models that are learned during exploration = problem-solving.

MC and TD often introduce a second (behavior) policy ($b$).
Then, $b$ handles exploration, while allowing $\Pi$ to exploit.
**On-Policy**

- **Target policy (Π) = Behavior policy (b)**
- Π determines moves for traversing search space (i.e. problem solving). Results of these moves influence changes to the value function.
- Exploitation easily becomes too dominant. Remedy: Π includes ε (random) moves.
- Early actions based on a weakly refined policy, Π.

**Off-Policy**

- Search (move) decisions and value-function updates are driven by the **behavior strategy (b);** they are independent of Π.
- Goal is still to improve Π.
- More explorative, trying to investigate all possible moves instead of just doing those that, so far, seem best for a given context.
- Early and late actions based on general quest to gain more knowledge about the environment and thus improve the value function.
On-Policy ... Off-Policy

**Learner**

- **State(t)**, **Action(t)**, **State(t+1)**, **Reward(t+1)**

- **Update**

**Target Policy** $\Pi$

- **Value Function**

- **State(t)**, **Evaluation**

- **Update**

**Agent**

- **State(t+1)**, **Reward(t+1)**

**Environment**

- **Action(t)**

**On-Policy RL**: $b = \Pi$

**Off-Policy RL**: $b \neq \Pi$

**Action Determined by Behavior Policy** ($b$)
Temporal Difference (TD) Learning

\[ V(s) \leftarrow V(s) + \alpha [r + \gamma V(s') - V(s)] \]

- \( \alpha \) = learning rate,
- \( \gamma \) = discount factor,
- \( r \) = reward after moving from state \( s \) to \( s' \)
- \( \delta = r + \gamma V(s') - V(s) = \text{TD Error} = \text{Level of Surprise} \)

This is 1-step TD, a.k.a. TD(0) \( \equiv \text{TD}(\lambda = 0) \); see TD(\( \lambda \)) below.
To update $V(S_t)$, combine discounted rewards from $t+1$ to $t+n$.

Add in $\gamma^n V(S_{t+n})$. This is bootstrapping: using estimated value of one state to update the estimated value of another.

$$G_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V(S_{t+n})$$

Then do standard (1-step) TD using $G_t$ as the target:

$$V(S_t) = V(S_t) + \alpha [G_t - V(S_t)]$$

Update target policy ($\Pi$) to be $\varepsilon$-greedy w.r.t. $V(S)$. This is an on-policy approach, so $\Pi$ needs to be $\varepsilon$-greedy to maintain some exploration.

If episode length = $N$, then $N$-step TD $\equiv$ MC.
Similar to n-Step TD, and also on-policy, but now use Q values:

- To update $Q(S_t, A_t)$, combine discounted rewards from $t+1$ to $t+n$.
- Add in $\gamma^n Q(S_{t+n}, A_{t+n})$. This is **bootstrapping**: using estimated value of one state-action pair to update the estimated value of another.

$$G_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n})$$

- Then do standard (1-step) TD using $G_t$ as the target:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha [G_t - Q(S_t, A_t)]$$

- Update target policy ($\Pi$) to be $\varepsilon$-greedy w.r.t. $Q(S,A)$. 

---

Keith L. Downing  Reinforcement Learning
### Q-Learning (Watkins, 1992)

**Off-policy TD using state-action pairs**

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

- **\(Q(s_t, a_t)\)** = Expected total remaining reward if performing action \(a_t\) in state \(s_t\).
- \(\gamma\) = discount rate
- \(\alpha\) = learning rate
- \(\delta = r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\) = TD Error = **Surprise**

This is **off-policy** due to the use of \(\max_a Q(s_{t+1}, a)\), which says to base the update on the **best possible** move from \(s_{t+1}\), not necessarily on the move sanctioned by the current version of the target policy.
Instead of a discrete number of backup steps, all states have an eligibility for update which gradually decays if the state is not visited during problem-solving search.

\[ e_t(s) = \begin{cases} 
\gamma \lambda e_{t-1}(s) & \text{if } s \neq s_t \\
\gamma \lambda e_{t-1}(s) + 1 & \text{if } s = s_t 
\end{cases} \]

- $s_t = \text{state of system at the current time, } t.$
- $\gamma = \text{discount rate}$
- $\lambda = \text{trace-decay factor}$

Now, instead of just the last $k$ (visited) states, ALL states (in the current episode) are updated, although many by a very small amount.
For each step of a problem-solving episode, do:

1. \( a \leftarrow \text{the action with highest probability in the policy component } \Pi(s) \)
2. Performing action \( a \) from state \( s \) moves the system to state \( s' \) and achieves the immediate reinforcement \( r \).
3. \( \delta \leftarrow r + \gamma V(s') - V(s) \) (TD Error)
4. \( e(s) \leftarrow e(s) + 1 \)
5. \( \forall s \in S \)
   - \( e(s) \leftarrow \gamma \lambda e(s) \) (Eligibility Decay)
   - \( V(s) \leftarrow V(s) + \alpha \delta e(s) \) (Value Update)

Similar for Q-Learning, but with:

- \( \delta = r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \) (TD Error)
- \( Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a) \) (Value Update)
TD($\lambda$) -vs- TD(2)

TD (2 step)

TD with:
- eligibility decay rate = 0.9
- discount rate = 0.9
- learning rate = 1.0

Start

Goal

Keith L. Downing
Reinforcement Learning
Critic handles value function and its updates, based on TD-error.

Actor handles the policy and its updates, but based **only** on TD error, not the value function (as in DP, MC, SARSA and Q-Learning).
Transition probabilities modified by TD error ($\delta$) and eligibility trace, $e(s,a)$:

$$\Pi(s,a) \leftarrow \Pi(s,a) + \alpha \delta e(s,a)$$

Transition probabilities normalized:

$$\Pi(s,a) \leftarrow \frac{\Pi(s,a)}{\sum_{a' \in \Pi(s)} \Pi(s,a')}$$

In contrast to the Generalized Policy Iterator (GPI), the actor-critic policy updates are only indirectly based on $V(s)$, since the actor only receives $\delta$, while the critic computes $\delta$ from $V(s)$ and $V(s')$. 
Ackley and Littman’s (1992) Actor-Critic ANN Pair

- Initial weights for both NNs are evolved.
- Only actor net learns (via a version of backprop), with help from outputs of the critic.

Keith L. Downing

Reinforcement Learning

1. Temporal differencing is the most feasible approach to state and policy updating, due to:
   - The enormous search-space size of typical ANN problem domains, which make comprehensive updates very costly.
   - The difficulty of wiring up neural circuitry to handle DP or MC approaches.

2. Eligibility traces are a simple bookkeeping mechanism requiring no extra caches (of, for example, all states in the current episode) nor (temporary) rewiring of the network.

3. The actor-critic separation of $V(s)$ from $\Pi(s)$ maps nicely to modular neural networks, whereas a tightly integrated combination of the two does not.

4. The basal ganglia of the mammalian brain shows actor-critic structure.
On the surface, RL looks hopelessly resource consuming, since many trials are required to gain and backup information about each state.

Monte Carlo methods and TD learning greatly simplify things.

TD methods allow a system to bootstrap its way to intelligence.

This was a key breakthrough in RL applicability to complex problems.