Reinforcement Learning

Greg Grudic

Reinforcement Learning (RL)

Autonomous agent learns without human intervention

- Agent learns by stochastically interacting with its environment, getting infrequent rewards
- Goal: maximize reward

Reinforcement Learning

- Addresses the temporal credit assignment problem:
  - Delayed reward (HARD problem!)
- Some successful RL applications:
  - TD gammon (Tesauro)
  - Packing containers (Moore)
  - Elevator dispatch (Crites and Barto)

Reinforcement Learning Problem

Markov Decision Processes

Assume
- Finite set of states \( S \)
- Set of actions \( A \)
- At each discrete time, agent observes state \( s_t \in S \) and chooses action \( a_t \in A \)
- Then receives immediate reward \( r_t \)
- And state changes to \( s_{t+1} \)
- Markov assumption: \( s_{t+1} = \delta(s_t, a_t) \) and \( r_t = c(s_t, a_t) \)
  - I.e., \( r_t \) and \( s_{t+1} \) depend only on current state and action
  - Functions \( c \) and \( \delta \) may be nondeterministic
  - Functions \( c \) and \( \delta \) not necessarily known to agent

Agent’s Learning Task

Execute actions in environment, observe results, and:
- Learn action policy \( \pi: S \rightarrow A \) that maximizes
  \[ \sum_{t=0}^{\infty} \gamma^t r_{t+1} \]
  from any starting state in \( S \)
- Here \( 0 \leq \gamma < 1 \) is the discount factor for future rewards

Different from supervised learning:
- Target function \( c: S \rightarrow A \)
- But we have no training examples of form \( (s, a) \)
- Training examples are of form \( (s, a, r) \)
Value Function

To begin, consider deterministic worlds...

For each possible policy π the agent might adopt, we can define an evaluation function over states

\[ V^π(s) = r(s, a) + \gamma V^π(δ(s, a, a)) \]

where \( s, a, \ldots \) are generated by policy \( π \) starting at state \( s \)

Rmined, the task is to learn the optimal policy \( π^* \)

\[ π^* ⊆ \text{argmax}_π V^π(s, π(s)) \]

What to Learn

We might try to have agent learn the evaluation function \( V^∗ \) (which we write as \( V^π \))

It could then do a look-ahead search to choose best action from any state \( s \) because

\[ π^*(s) = \text{argmax}_a [r(s, a) + \gamma V^π(δ(s, a, a))] \]

A problem:

- This works well if agent knows \( δ : S × A → S \) and \( r : S × A → \mathbb{R} \)
- But when it doesn’t, it can’t choose actions this way

Q Function

Define new function very similar to \( V^∗ \)

\[ Q(s, a) = r(s, a) + \gamma V^π(δ(s, a, a)) \]

If agent learns \( Q \), it can choose optimal action even without knowing \( δ \)

\[ π^*(s) = \text{argmax}_a [r(s, a) + \gamma V^π(δ(s, a, a))] \]

\[ π^*(s) = \text{argmax}_a Q(s, a) \]

\( Q \) is the evaluation function the agent will learn

Training Rule to Learn \( Q \)

Note \( Q \) and \( V^∗ \) closely related:

\[ V^∗(s) = \max_a Q(s, a') \]

Which allows us to write \( Q \) recursively as

\[ Q(s, a) = r(s, a) + \gamma \max_{a'} Q(δ(s, a, a')) \]

Nice! Let \( Q \) denote learner’s current approximation to \( Q \). Consider training rule

\[ Q(s, a) → r + \gamma \max_{a'} Q(δ(s, a, a')) \]

where \( a' \) is the state resulting from applying action \( a \) in state \( s \)

Q Learning for Deterministic Worlds

For each \( s, a \) initialize table entry \( Q(s, a) → 0 \)

Observe current state \( s \)

Do forever:

- Select an action \( a \) and execute it
- Receive immediate reward \( r \)
- Observe the next state \( s' \)
- Update the table entry for \( Q(s, a) \) as follows:

\[ Q(s, a) → r + \gamma \max_{a'} Q(δ(s, a, a')) \]

- \( s → s' \)
**Updating Q**

**Initial state:** \( s_1 \)

**Next state:** \( s_2 \)

\[ Q(s, a_{opt}) = r + \gamma \max_{a'} Q(s', a') \]

\[ = r + \gamma \max_{a'} Q(s', a') \]

\[ = r + \max_{a'} [r(s', a') + \gamma Q(s', a')] \]

\[ = r + \max_{a'} [r(s', a') + \gamma Q(s', a')] \]

\[ = 0.9 \cdot \max_{a'} [r(s', a') + \gamma Q(s', a')] \]

\[ \leq 0.9 \cdot \max_{a'} [r(s', a') + \gamma Q(s', a')] \]

\[ \Delta = Q(s, a_{opt}) - Q^*(s) \]

**Proof:** Define a full interval to be an interval during which each \( (s, a) \) is visited. During each full interval the largest error in \( Q \) table is reduced by factor of \( \gamma \).

Let \( \Delta \) be the maximum error in \( Q \) at \( s \) that is:

\[ \Delta = \max_a (Q(s, a) - Q^*(s)) \]

For any full interval \( Q(s, a) \) updated on iteration \( e \), the error in the revised estimate \( Q_{e+1}(s, a) \) is:

\[ |Q_{e+1}(s, a) - Q^*(s)| \leq |r + \gamma \max_{a'} Q(s', a') - \gamma Q^*(s)| \]

\[ \leq |r + \gamma \max_{a'} Q(s', a') - \gamma Q^*(s)| \]

\[ \leq \gamma |\max_{a'} [r(s', a') + \gamma Q(s', a')] - \gamma Q^*(s)| \]

\[ \leq \gamma |\max_{a'} [r(s', a') + \gamma Q(s', a')] - \gamma Q^*(s)| \]

Note we used general factor:

\[ |\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)| \]

**Nondeterministic Case**

What if reward and next state are non-deterministic?

We redefine \( V, Q \) by taking expected values:

\[ V^*(s) = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots] \]

\[ = E[\sum_{t=0}^{\infty} \gamma^t r_{t+1}] \]

\[ Q(s, a) = E[V(s, a) + \gamma V^*(s, a)] \]

**Temporal Difference Learning**

Q-learning: reduce discrepancy between successive Q-estimates

One step time difference:

\[ Q^t(s, a) = r + \gamma \max_{a'} Q(s', a') \]

Why not two-steps?

\[ Q^t(s, a) = r + \gamma r_{t+1} + \gamma^2 \max_{a'} Q(s', a') \]

Or \( a' \):

\[ Q^t(s, a_{opt}) = r + \gamma r_{t+1} + \gamma^2 \max_{a'} Q(s', a') \]

Blend all of these:

\[ Q^t(s, a) = (1 - \lambda) Q^t(s, a) + \lambda Q^t(s, a) + \lambda^2 Q^t(s, a) + \ldots \]

**Temporal Difference Learning**

Q-learning generalizes to nondeterministic worlds

After training rule to:

\[ Q_{e+1}(s, a) = Q_e(s, a) - \gamma \alpha [r + \gamma \max_{a'} Q_{e+1}(s', a') - Q_{e+1}(s, a)] \]

where:

\[ \alpha_e = \frac{1}{1 + \gamma \alpha_e} \]

Can still prove convergence of \( Q \) to \( Q \) [Watkins and Dayan, 1992]

**Nondeterministic Case**
RL Application Domains

**Successful domains**
- Low dimensional discrete state space
- 1,000,000’s learning runs \(\text{(simulation)}\)

**Not so successful domains**
- Large continuous state space
- 1,000,000’s learning runs *not* practical

---

Continuous Domains? Robotics

- Hit an obstacle: get a **negative** reward
- Reach goal: get a **positive** reward
- Reach goal faster: get a **bigger positive** reward

---

A (simple?) Robotics Problem

Even Simple tasks are difficult to Program

---

Other Continuous Problems

- Process control
  - Chemical
  - Power
- Financial modeling
- Software agents on web
  - State space defined by hit statistics
- TCOM problems, etc

Why Is RL Hard in Large Continuous Domains?

- Stochastic search in large continuous domains is hard

  *One possible solution: use prior domain knowledge to direct search*
Reinforcement Learning (MDP)

- Policy: \( \pi(s,a) = \Pr(a_s = a|s = s) \)
- Reinforcement feedback (environment) \( r_t \)
- Goal: modify policy to maximize reward
  \( \rho(\pi) = E\left[ \sum_{t=0}^{\infty} r_t | s_0 = s, \pi \right] \)
- State-action value function
  \( Q^\pi(s,a) = E\left[ \sum_{t=0}^{\infty} r_t | s_0 = s, a_0 = a, \pi \right] \)

Approaches to RL

- Value Function RL
- Policy Gradient RL
- Actor-Critic RL
  – combines value functions and policy gradients

Value Function RL

- Learn the value of executing each action in each state (Value Function \( Q^\pi(s,a) \))
- In each state, execute the most valuable action
- Problem:
  – Value function learning infeasible in high dimensional state spaces

Policy Gradient RL

- Parameterize agent’s policy (\( \Theta \))
- Estimate how the value of a policy (\( \rho \)) changes as \( \Theta \) changes: \( \frac{\partial \rho}{\partial \Theta} \)
- Update policy (gradient ascent):
  \( \Theta_{t+1} = \Theta_t + \alpha \frac{\partial \rho}{\partial \Theta} \)
- Problems:
  – Local minimum and slow convergence

The Better Approach for large domains: Policy Gradient RL

Why?

- Globally optimal solutions are intractable
- Computational cost of a performance gradient estimate is linear with size of state space
  – Value functions have exponential growth
- Prior knowledge directly encoded in policy parameter vector

PGRL Algorithms

- Agent uses a stochastic policy
  \( \pi(s,a|\Theta) = \Pr\{a = a|s; \Theta\} \)
  \( \Delta \Theta = \frac{\partial \rho}{\partial \Theta} = f\left( \frac{\partial \pi}{\partial \Theta}, Q^\pi(s,a) \right) \)
  \( \frac{\partial \pi}{\partial \Theta} \) Must exist and be bounded

Williams 1987, 1992; Baird and Moore 1999; Baxter and Bartlett 2000; Sutton, McAllester, Singh, Mansour, 2000; Konda and Tsitsiklis 2000
Implications of Stochastic Policy

- Use stochastic exploration during learning
- Stochastic exploration can be expensive in large state spaces
  - High variance in performance gradient estimate
- Must direct search during learning!

Performance Gradient Estimates

- **Need relative values of executing actions in state**
- Stochastic PG algorithms obtain absolute estimates
  - Relative estimates are indirectly (stochastically) obtained after many visits to the same state
  - poor sampling technique

Action Transition Policy Gradient (ATPG)

- Policy Gradient estimates restricted to when agent changes actions
- Gives a direct estimate of the *relative* value of executing actions
- **Theorem**: Convergence to locally optimal policies theoretically guaranteed (Grudic and Ungar ICML 2000)

ATPG Simulation Results

<table>
<thead>
<tr>
<th>Method</th>
<th>Average Episodes for Convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>REINFORCE</td>
<td>&gt; 1,000,000 (L=200)</td>
</tr>
<tr>
<td>PFA</td>
<td>240 (sd 340) (L=1)</td>
</tr>
<tr>
<td>ATPG</td>
<td>230 (sd 30) (L=3)</td>
</tr>
</tbody>
</table>

Boundary Localized Reinforcement Learning (BLRL)

- Transform a stochastic policy into one that is deterministic everywhere except near mode boundaries
- Any stochastic policy that generalizes in state can be transformed to BLRL
  - Parameters shared among states

Advantages of BLRL

- **Theorem**: Convergence to locally optimal mode switching policies is obtained by searching near mode boundaries (Grudic and Ungar, AAAI 2000)
- Most of the state space can be ignored when estimating a performance gradient
BLRL Search Region

2-D Simulation

Convergence in Higher Dimensions

Controllers are Typically Deterministic

Mode Examples From Robotics

Mode Switching Controllers
**Other Controller Paradigms: Action Superposition Controllers**

- Deterministic continuous action space
  \[ a_j = \sum_{i=1}^{M} a_i g_i(x, \Theta) \]
- Used in Potential Field Methods
  - Path planning

**Deterministic Policy Gradient (DPG)**

- Most robot controllers are deterministic
- Stochastic PG algorithms will not directly work with these controllers
  - E.G. \( \frac{d\pi}{d\Theta} \) is infinite
- Propose a deterministic PG formulation

**Deterministic Perturbations in Policy Space**

Goal
Robot
Obstacle
Static Navigational Feature

**Which Policy Parameters are Important?**

- Deterministic policy defined by \( M \) functions
  \[ g_1(s(t), \Theta), \ldots, g_M(s(t), \Theta) \]
- Theorem:
  \[ \frac{d\rho}{d\theta_k} = \int \left( \sum_{i=1}^{M} \frac{\partial g_i(s,t)}{\partial \theta_k} \frac{\partial g_i(s,t)}{\partial \theta_k} \right) dt \]
  - Can ignore all \( \theta_k \) when \( \frac{d\rho}{d\theta_k} \) is small

**Deterministic Policy Gradient Algorithm**

1. Estimate which parameters are significant under the current policy
2. Systematically perturb these parameters and estimate a performance gradient
3. Update the policy in a direction of increased reward
4. GOTO Step 1

**Action Superposition Simulation**

Initial | Learned
AS Convergence Results

Mode Switching Simulation

MS Convergence Results

Does this work on Actual Robots?

Yes!

Rate of Convergence Results

- ATPG
- BLRL
- DPG
Other Types of Prior Knowledge

- Secondary Reinforcers
- Control the relative amounts of these (Grudic and Ungar IJCAI 01)

Conclusions

- Policy Gradient Framework in RL effective in large problem domains:
  - Selective sampling of the state space (ATPG)
  - BLRL reduces search to mode boundaries
  - Deterministic policy perturbations (DPG) give effective performance gradient estimates for deterministic controllers
- However
  - Prior domain knowledge is required (and easily incorporated!)
  - Globally optimal solutions are not learned