MACHINE INTELLIGENCE COMMUNITY

Summary of Reinforcement Learning: An Introduction Chapter 1: Introduction

Justin Chen* chenjus@bu.edu
Tyrone Hou* tyroneh@bu.edu

January 15, 2018

1 Introduction

- **Reinforcement Learning** - Mapping situations to actions to maximize a **scalar** reward
- Defining characteristics of reinforcement learning problems:
  - **Trial-and-error search**
  - **Delayed reward**
  - Learning from interaction
    - Observe state of environment
    - Perform actions that affect state of environment
- Reinforcement Learning v. Supervised Learning
  - Supervised learning requires correctly labeled information that covers all possible scenarios
  - Usually not possible to obtain such a thorough and representative set of data and agent must learn from experience
- Reinforcement Learning v. Unsupervised Learning
  - Unsupervised learning aims to find hidden structure
  - Reinforcement learning aims to maximize a reward
- **Exploration v. Exploitation**
  - Agent must balance two strategies
  - **Exploit**: repeat its current strategy to maximize reward
  - **Explore**: try something other than what the agent already knows to possibly find a better strategy
  - Unsolved problem
  - May require agent multiple tries to reliably estimate expected reward
- Reinforcement learning problems consider the entire problem and subproblems of the goal-directed agent in an uncertain environment.
- Agents:
  - Have explicit goals
  - Can observe environment
  - Take actions to influence the environment

*These authors contributed equally to this work

1.2 Examples

- Agent's actions affect the future state of the environment which affect the available actions and states in the future
- Correct actions require considering indirect, delayed consequences, which require foresight and planning
- Examples: Playing go, flying a drone, driving to work, etc.

1.3 Elements of Reinforcement Learning

- **Agent**
  - Goal seeking entity
- **Environment**
  - Everything outside that agent interacts with
  - Agent can have *partial information* about environment
- **Policy**
  - Agent's behavior given a situation
  - Mapping between states and actions
  - Stochastic: $P(a_i|s_j)$ of taking action a_i in state s_j
  - Deterministic: Take action a_i in state s_j

$$State \rightarrow Policy \rightarrow Action$$

- **Reward Signal**
  - *Immediate* signal of how good an action is
  - Scalar value
- **Value function**
  - Function predicting *expected future reward* for a state
  - Estimated by taking actions and receiving rewards over time
  - Agent seeks to optimize *long-term* value, not *immediate* reward

$$State \rightarrow Value \rightarrow ExpectedReward$$

- **Environment model** (optional)
  - Model that simulates behavior of environment for the agent

$$\text{Model-free} \xleftarrow[\text{Trial-and-error}]{\qquad\qquad} \xrightarrow[\text{Planning}]{} \text{Model-based}$$

1.4 Limitations and Scope

- Reinforcement learning relies on state: "how the environment is" at a given time
  - State representation is important to solving the RL problem (See section 17.3)
- Methods of solving reinforcement learning problem
  - **Value function estimation**
    - Does not directly model policy space

    - Computes policy with value function between states and actions
  - **Evolutionary search**
    - Modeled after biological evolution
    - Agents are *non-learning*
    - Samples many static policies and evaluates "lifetime" reward of each one
    - Chooses policy with most total reward
    - e.g. Genetic algorithms, genetic programming, simulated annealing
  - **Gradient Descent**
    - Models policy space as a collection of numerical parameters
    - Finds direction of steepest ascent (in terms of reward)
    - Best GD methods *usually* include value functions

- Differences between methods
  - Value functions account for structure of RL problem:
    - Models how individual states and actions affect reward
  - Evolutionary methods *do not interact* with environment
    - Only look at how the policy performs over its lifetime
  - Evolutionary methods can be effective when
    - Only *partial information* about state/environment is available
    - Policy space is *small*

1.5 An Extended Example: Tic-Tac-Toe

- Assume we are playing tic-tac-toe with an **imperfect player**
- How can we define an agent that capitalizes on opponent's mistakes to win more often?
- Non-reinforcement Learning approaches:
  - Classical minimax solution
    - Computes optimal solution assuming an *optimal opponent*
    - Computes optimal solution assuming *any opponent*
    - Requires complete description of opponent behavior
  - We often don't have complete knowledge of how the opponent likes to play
- Reinforcement learns a model of opponent's behavior by playing many games before trying to compute an optimal action
- Characteristics of reinforcement learning approaches for tic-tac-toe
  - Discrete state and action Space
    - States are all possible board states with checker positions
    - Actions are all places an agent can place a checker
  - Trial-and-error search
    - Play many games against opponent
  - Learning through interaction
    - Update value function throughout each game
  - Delayed rewards
    - Agent doesn't get a reward until a game ends
  - Exploration v. Exploitation
    - Agent selects optimal action most of the time
    - With low probability, agent chooses a random action

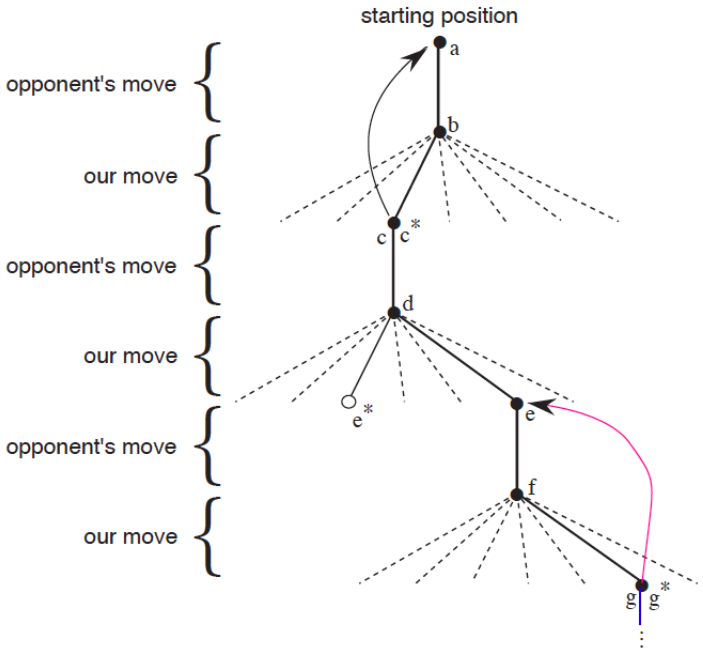

Figure 1.1: A sequence of tic-tac-toe moves. The solid lines represent the moves taken during a game; the dashed lines represent moves that we (our reinforcement learning player) considered but did not make. Our second move (d-e) was an exploratory move, meaning that it was taken even though another sibling move, the one leading to e_*, was ranked higher. Exploratory moves do not immediately result in any learning, but subsequent moves that are optimal do, causing updates as suggested by the curved arrow in which estimated values are moved up the tree from later nodes to earlier as detailed in the text.

  - Credit assignment
    - For a given board (state) *more value* is given to actions that result in a next state with higher value

- Estimate updates
  - Make more accurate state value estimates over time
  - **Temporal-difference learning** - Update value of earlier states to closer approximate value of later states
    - Similar idea to backpropagation and gradient ascent
    - α is **step-size**/ learning rate
    - $V(s)$ is value of state s

$$V(s) \leftarrow V(s) + \alpha\big[V(s') - V(s)\big]$$

- Beyond Tic-Tac-Toe
  - RL can also be applied to more complex situations
  - Very large discrete or even **continuous** state and action spaces
    - e.g. Backgammon with 10^{20} states (Richard Tesauro, 1992, 1995)
  - No external adversary exists: "Game against nature"
  - Partial state information
  - Hierarchical planning system
    - Can chain a 'low-level' move decider with a 'high-level' strategy subsystem to create a complete RL system

1.6 Summary

- Reinforcement learning is a framework for goal-directed learning and decision making
- Agent must interact with environment to learn
- Value functions are key for efficiently searching policy space

1.7 Early History of Reinforcement Learning

- Three historical approaches towards reinforcement learning:
  - Learning by trial-and-error from psychology
  - Optimal control problem
    - Late 1950s
    - Problem of designing controllers to minimize measures in a dynamic system's behavior over time
    - **Optimal return function**
      - Using states and value function to define a functional equation
      - Class of problems now known as **dynamic programming** coined by Richard Bellman in 1957
      - **Bellman equation** was developed Mid-1950s
    - Bellman introduced **Markovian Decision Processes (MDPs)** in 1957
      - Discrete stochastic version of the optimal control problem
    - Ron Howard developed the **policy iteration** method in MDPs in 1960
  - Temporal-difference methods
- Modern reinforcement learning started in late 1980s
- In 1996, Dimitri Bertsekas and John Tsitsiklis coined the term **neurodynamic programming** - using neural networks with dynamic programming
- **Law of Effect**
  - "The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond." (Edward Thorndike, 1911)
- First use of the term "reinforcement" appeared in 1927 in an English translation of Pavlov's monograph on conditioned reflexes
  - Reinforcement galvanizes different behavior that persists even after the reinforcer is no longer present
- Alan Turing was the first to suggest implementing reinforcement learning in machine
- Marvin Minsky described the **basic credit-assignment problem** in his 1961 paper *Steps Toward Artificial Intelligence*
  - Problem of how to distribute credit to many actions that may have been involved in producing a single reward
- Arthur Samuel was the first to propose and demonstrate temporal-difference learning in 1959 by developing a reinforcement learning agent to play checkers
- Harry Klopf was responsible for reviving the study of trial and error learning in 1972
- Chris Watkins developed Q-learning in 1989 which brought together ideas in optimal control and temporal-difference learning
- Gerry Tesauros developed a reinforcement learning agent to play backgammon in 1992

References

Sutton, Richard S., and Andrew G. Barto. “Introduction.” Reinforcement Learning: An Introduction, The MIT Press, 2018, pp. 1–17.