21: Reinforcement Learning

Reinforcement Learning (RL) – the task of using observed *rewards* to learn a (approximately) optimal policy for an environment by choosing an action that will maximize the *expected reward* given the current observed state of the agent.

- **Reward (Reinforcement)** – feedback that differentiates between *good* and *bad* outcomes; thus allowing the agent to make choices.
- RL builds on the studies of animal psychologists in differentiating between *reward* and other sensory inputs.
- Unlike MDPs, RL agents assume no prior knowledge of either the environment or the reward function.
- *In a sense, the RL task encompasses all of AI*: an agent is placed in an environment where it must behave successfully.
- Three types of agent designs:
  - ***utility-based agent*** – learn a utility function for states, which the agent will use to select actions in order to maximize expected utility.
    - requires an environment model to map actions to successor states.
  - ***Q-learning agent*** – learns a utility function on the state-action pairs; a so-called Q-function.
    - able to compare actions without knowing their outcomes.
    - without knowing action outcome, look ahead is not possible.
  - ***reflex agent*** – learns a policy that maps states to actions.

Passive Reinforcement Learning – the agent's has a fixed policy π: perform action $\pi(s)$ in state *s*. This is similar to *policy iteration*, but we lack the *transition model* $T(s,a,s')$ and the *reward function* $R(s)$. Thus, the agent performs a set of **trials** and uses the observed rewards to estimate the expected utility of each state $U^{\pi}(s)$. Starting in state *s* we want to estimate the (discounted) expected reward from future states:

$$U^{\pi}(s) = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0 = s\right]$$

- **Direct Utility Estimation** – the utility of a state is the expected reward starting from that state, so each **trial** is a sample for each state visited.
  - In this setting, the problem becomes a supervised learning problem of mapping state to value ➔ an inductive learning problem.
  - This *Monte-Carlo* approach assumes independence of the utility function between states. This ignores the fact that utilities are coupled in the Bellman equations! Thus, this approach does not **bootstrap**!
    - Without bootstrapping, invaluable information for learning is lost and thus the technique converges very slowly.

- **Adaptive Dynamic Programming (ADP)** – as the agent moves through the environment, the transition model is estimated and the MDP for the corresponding model estimate is solved incrementally using dynamic programming.
    - Learning the environment:
        - The transition model $T(s,a,s')$ is estimated from the frequency from state *s* to state *s'* via action *a*.
    - The MDP is solved using policy iteration or modified policy iteration.
    - ADP is intractable for large state spaces.
    - approximate ADP – bounds the number of adjustments per transition.
        - *prioritizing sweep heuristic* – prefers to adjust states whose successors have recently had a large utility adjustment.
- **Temporal Difference (TD) Learning** – a mixture of sampling and constraint bootstrapping in which the values of the observed states are modified to reflect the constraints between states given by the MDP.
    - TD equation: given a learning rate α we update the expected utilities:
    $$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha\left(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s)\right)$$
    The TD equations converges to the MDP equilibrium even though only visited states are considered – the frequency of visits to a state are a substitute for the explicit transition model.
    - TD is an efficient approximation of ADP:
        - the utility function is updated by local adjustments.
        - TD only adjusts w.r.t. the observed transition and only makes a single update per transition.

Active Reinforcement Learning – policy is no longer fixed; active agents must decide on actions to take.

- **Exploration**
    - ***greedy agent*** – follows the current "optimal policy" according to the current estimates of the utility of each state.
        - unlikely to converge to the "optimal policy" since neglected states have poor estimates of their utility functions.
    - Trade-off between exploration and exploitation
        - ***exploitation*** – utilizing current knowledge to perform actions that maximize rewards.
        - ***exploration*** – trying suboptimal actions with the hope of improving our current estimates for the utility function.
        - *n*-armed bandit – a slot machine with *n*-levers – gambler must choose to exploit the lever with highest payoff or explore other levers to better estimate their payoff.
            - *Gittins index* – a measure of this tradeoff in independent situations (doesn't extend to sequential decisions).

- **Greedy in the limit of infinite exploration (GLIE**) – exploration schemes that are eventually optimal.
  - simple GLIE scheme – try a random action with probability $1/t$; otherwise, perform the optimal action.
  - optimistic utility estimates that favor unexplored states:

$$U^+(s) = R(s) + \gamma \max_a f\left(\sum_{s'} T(s,a,s')U^+(s'), N(a,s)\right)$$

    - U^+ is the optimistic utility function
    - $N(a,s)$ is the # of times action *a* is done in state *s*.
    - *exploration function* $f(u,n)$ - trade-off between greed and curiosity that must increase in *u* and decrease in *n*. e.g.

$$f(u,n) = \begin{cases} R^+ & n < N_e \\ u & otherwise \end{cases}$$

    - policy converges quickly while utility estimates converge slowly, but all we need is correct policy!

- **Action-Value Function**
  - *TD-learning* can be adapted to the active setting simply by choosing an action based on the current *U* estimate via 1-step look-ahead. However, we still have to learn the environment model to select actions.
  - **Q-learning** – learns an action-value representation instead of utilities.
    - Q-values: $U(s) = \max_a Q(a,s)$
    - ***model-free*** – does not require an environment model for learning or action selection.
    - *Bellman equations for Q-values*:

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

    - *TD Q-learning*: (model-free)

$$Q(a,s) \leftarrow Q(a,s) + \alpha\left(R(s) + \gamma \max_{a'} Q(a',s') - Q(a,s)\right)$$

      - TD doesn't enforce consistency between values by using the model so it learns slower!
  - **knowledge-based approach** – method of representing the agent function by building a model of some aspects of the agent's environment.
    - Has definite advantages over model-free learning agents as the environment becomes more complex.

Generalization in Reinforcement Learning – we now consider methods for scaling RL to worlds with enormous state spaces. Standard tabular RL is impractical since the table has one entry per state and since most states would be visited rarely.

- **function approximation** - representing the value function in (approximate) non-tabular forms, e.g., a linear combination of ***features*** of the state:

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \ldots + \theta_n f_n(s)$$

  - Thus we want to learn the parameters $\theta_1, \ldots, \theta_n$ to best approximate the value function. *Note*: features can be non-linear in the state variables.
  - *Function approximation allows the agent to broadly generalize between many states via states' common attributes.*
  - Unfortunately, the best utility function may be a poor estimate!
  - Online learning updates (**Widrow-Hoff** or **Delta Rule**): uses derivatives of squared error to update parameters.

$$\theta_i \leftarrow \theta_i + \alpha\left(u_j(s) - \hat{U}_\theta(s)\right)\frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$$

    - TD update: $\theta_i \leftarrow \theta_i + \alpha\left(R(s) + \gamma \hat{U}_\theta(s') - \hat{U}_\theta(s)\right)\frac{\partial \hat{U}_\theta(s)}{\partial \theta_i}$
    - Q: $\theta_i \leftarrow \theta_i + \alpha\left(R(s) + \gamma \max_{a'} \hat{Q}_\theta(a', s') - \hat{Q}_\theta(a, s)\right)\frac{\partial \hat{Q}_\theta(a, s)}{\partial \theta_i}$

  - These updates converge to the optimal estimate for linear functions, but can wildly diverge for non-linear ones.
- Function approximation can also be used to estimate the environment model:
  - in *observable models*, this is a supervised task.
  - in *partially-observable* models, DBNs with latent variables can be used.