# Reinforcement Learning David Silver: Lecture 3 Notes

Name: Eli Andrew

- Dynamic programming
  - Dynamic sequential or temporal component to the problem
  - Program as in the policy we are trying to optimize
  - Method for solving complex problems by breaking into subproblems and then solving the subproblems and putting them together to arrive at a solution
  - Two necessary properties: (1) optimal substructure (problem can be broken into subproblems that can then be combined again) (2) overlapping subproblems (the subproblems occur many times)
  - MDPs have these two properties
    * Recursive decomposition is given by the Bellman Equation
    * Subproblem value caching can be acheived with the value function (stores all useful computed information about the MDP)
  - Assumes full knowledge of MDP and is used for planning
  - Can be used for prediction:
    * Input: MDP $< S, A, P, R, \gamma >$ and a policy $\pi$
    * Output: value function $v_\pi$
  - Can also be used for control:
    * Input: MDP $< S, A, P, R, \gamma >$
    * Output: optimal policy $\pi_*$ and optimal value function $v_*$
- Policy evaluation
  - This is when you are told the MDP and the policy and you want to calculate the value of the policy $v_\pi$
  - General strategy: perform an iterative application of the Bellman expectation backup
  - Start with inital state values in $v_1$ and then apply Bellman expectation to get $v_2$ and continue to converge to $v_p i$
  - Using synchronous backups:
    * At each iteration $k + 1$
    * For all states $s \in S$:
    * Update $v_{k+1}(s)$ from $v_k(s')$ where $s'$ is a successor state of $s$
    * Where $v_{k+1}(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s'))$
- Policy iteration

- Goal is to find the best possible policy in the MDP vs. evaluating a fixed policy like we did in policy evaluation
- One way to look at this is given a policy $\pi$ how can we return a polciy $\pi'$ that is better than $\pi$
- So, given a policy $\pi$
  - **Evaluate** the policy $\pi$: $v_\pi(s) = E[R_{t+1} + \gamma R_{t+2} + \ldots | S_t = s]$
  - **Improve** the policy by acting greedily with respect to $v_\pi$: $\pi' = greedy(v_\pi)$
- Policy iteration always converges to the optimal policy $\pi_*$
- On each iteration of this we generate a value function $v_\pi$ which we act greedily on to get some new policy $\pi'$, which on the next iteration gives a new value function $v_{\pi'}$ and then a new policy $\pi''$ and so on and so on until we obtain $v_*$ and $\pi_*$
- Put another way:
  - Consider a deterministic policy: $a = \pi(s)$
  - We can improve the policy by acting greedily: $\pi'(s) = argmax_{a \in A} q_\pi(s, a)$
  - Acting greedily at least improves value for a single step: $q_\pi(s, \pi'(s)) = max_{a \in A} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$
  - Follow this logic to the rest of the steps and you can assume its better for the whole trajectory
  - When improvement stops we have the condition: $q_\pi(s, \pi'(s)) = max_{a \in A} q_\pi(a, s) = q_\pi(s, \pi(s)) = v_\pi(s)$
  - This then satisfies the Bellman Optimality Equation