- CME 241 Notes : Reinforcement Learning for Predictions (17 - Feb - 2021)
  - ↪ No more required readings beyond this chapter
  - ↪ Module 3 : Reinforcement Learning

- DP/ADP assume access to a probability model (knowledge of P_R)
  - ↪ But if you know probabilities, you don't need to interact with environment
- In RL, we don't have access to a probability function
  - ∴ You must interact with actual environment
    - ↪ In state, make move, and environment produces one instance of next state and reward (individual Experiences)
    - ↪ Often you try to build a guessed P_R function by Supervised learning

- Rather than trial and error, you may want to fit a model by supervised learning.
  - ↪ The real world is not stationary
  - ↪ Models must be constantly updated
  - ↪ Also, State space is huge (curse of dimensionality)
  - ↪ Curse of Modeling : Probability estimation on large state space is intractable
    - ↪ It's difficult to stitch together individual probability models
    - * It's more practical to stitch Samples together
      - ↪ This is much more reasonable. Gives sampling model for P_R, which serves as a proxy for the real world.
      - ↪ Let RL interact with Simulated environment
        - ↪ Allows you to run over many experiences

- RL Needs an interface to Sample Experiences
  - ↳ RL learns the value function w/o access to probability, but rather by accessing a stream of simulated experiences
  - ↳ Under the right settings, RL will converge on value function
  - ↳ Like real life, RL does not attempt to build a probability model.
    - ↳ RL is a trial and error approach to linking actions to accumulated rewards (returns).
      - ↳ How do you disentangle actions from eventual returns?
      - ↳ Actions have highly overlapping reward sequences.
      - ↳ Some actions result in delayed reward.

- Our attention will shift to the Q-value function
  - ↳ Expected return conditioned on state and action.
  - ↳ RL constantly updates Q-value function
    - ↳ Supposed to be an expectation. Updated through new samples
    - ↳ Updates done through function approximation
      - ↳ Linear function
      - ↳ Neural Network
    - ↳ Almost all of RL is based on Bellman Equations

- Prediction is a stepping stone to control
  - ↳ Today and Friday we will talk about RL for prediction

- Prediction: Evaluate MDP with fixed policy π using individual experiences

  MDP + π = MRP

  ↳ We access samples from MRP to estimate value function

  $S_0 \longrightarrow R_1, S_1 \longrightarrow R_2, S_2 \longrightarrow \ldots$

  "Atomic Experience"

  "Trace Experiences"

  (S, R, S') ≡ TransitionStep.

- Software Design: Interface can be an iterable of atomic experiences

  ↳ Or you could have a stream of trace experiences by stitching together transition steps.

  ↳ Iterable can be either a Sequence or an Iterator (stream)

- Monte Carlo Prediction

  - You can get many sample returns for a state and you can fit a model with supervised learning.

    ↳ Implemented with incremental manner in FunctionApprox update() method.

    ↳ You can only do an update at the end of the trace experience because we are talking about returns.

    ↳ Returns are easier to calculate backwards

    ↳ Read "walk-back" code

↳ Minimizes MSE of predicted and observed return

↳ "Episode" is a finite trace experience

↳ MC Method requires every trace experience to end.

- Code:

"trs" is a stream of episodes

Specify an initial estimate of the initial function approximation

- Prediction output produces estimate of updated function approximators using a generator
- Function approx. iterate + updates ( ) calls update again and again. Sequence of updates for all episodes.
- Update Formula:

$$\Delta \underline{w} = \alpha \left(G_t - V(S_t ; w) \right) \cdot \nabla_w V(S_t ; w)$$

α: Learning rate

$G_t - V(S_t ; w)$: Return residue

$\nabla_w V(S_t ; w)$: Gradient of Expected returns

- Tabular MC Prediction
  - Denote $V_n(s_i)$ as the estimate of VF for first n occurrences of S_i

$$V_n(s_i) = V_{n-1}(s_i) + f(n) \cdot \left(Y_i^{(n)} - V_{n-1}(s_i) \right)$$

$f(n)$: Count to weight function

↳ One default approach is $f(n) = \frac{1}{n}$

↳ With this approach, the update function is a running average

↳ For tabular MC, incremental averaging done separately for each state

↳ Tabular MC does incremental average

↳ Law of large numbers ⇒ Average approaches true expectation

↳ Tabular MC is a specialized case of linear function approx

↳ Feature matrix is an n-dimensional identity matrix

↳ Weight vector is average of each state.

∴ $Y_i^{(n)} - W_i^{(n-1)}$ is the gradient of the loss function

- In general, you want to use a time-decaying weighted average of returns.
  - ↳ Lower weights for older data.

- This version of MC is called "Each Visit MC"
  - ↳ Update for each occurrence of a given State
  - ↳ Opposed to "First Visit MC" : Update only for first visits

- MC produces unbiased estimates, but it can be slow to converge
- MC requires complete trace experiences
  - ↳ The inability to use incomplete experiences is a key disadvantage

- Temporal Differences Prediction : Begin with a tabular view
  - ↳ TD Exploits recursive structure of VF in Bellman Equation
    - ↳ Replace G_t with $R_{t+1} + \gamma V(S_{t+1})$ using atomic experiences
      - ↳ This is a bootstrapping approach
  - ↳ Update to VF happens after every experience

$R_{t+1} + \gamma V(S_{t+1})$: "Temporal Difference Target"

$\delta = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$: "Temporal Difference Error"

$$V(S_t) \longleftarrow V(S_t) + \alpha \gamma$$

↳ Key idea: Unlike MC, TD is non-episodic

↳ We may have continuous traces $\Rightarrow \gamma < 1$

↳ TD can be run on any stream of atomic experiences

$\therefore$ You can split up / rearrange atomic experiences

- Switching to TD Prediction with Function Approximation