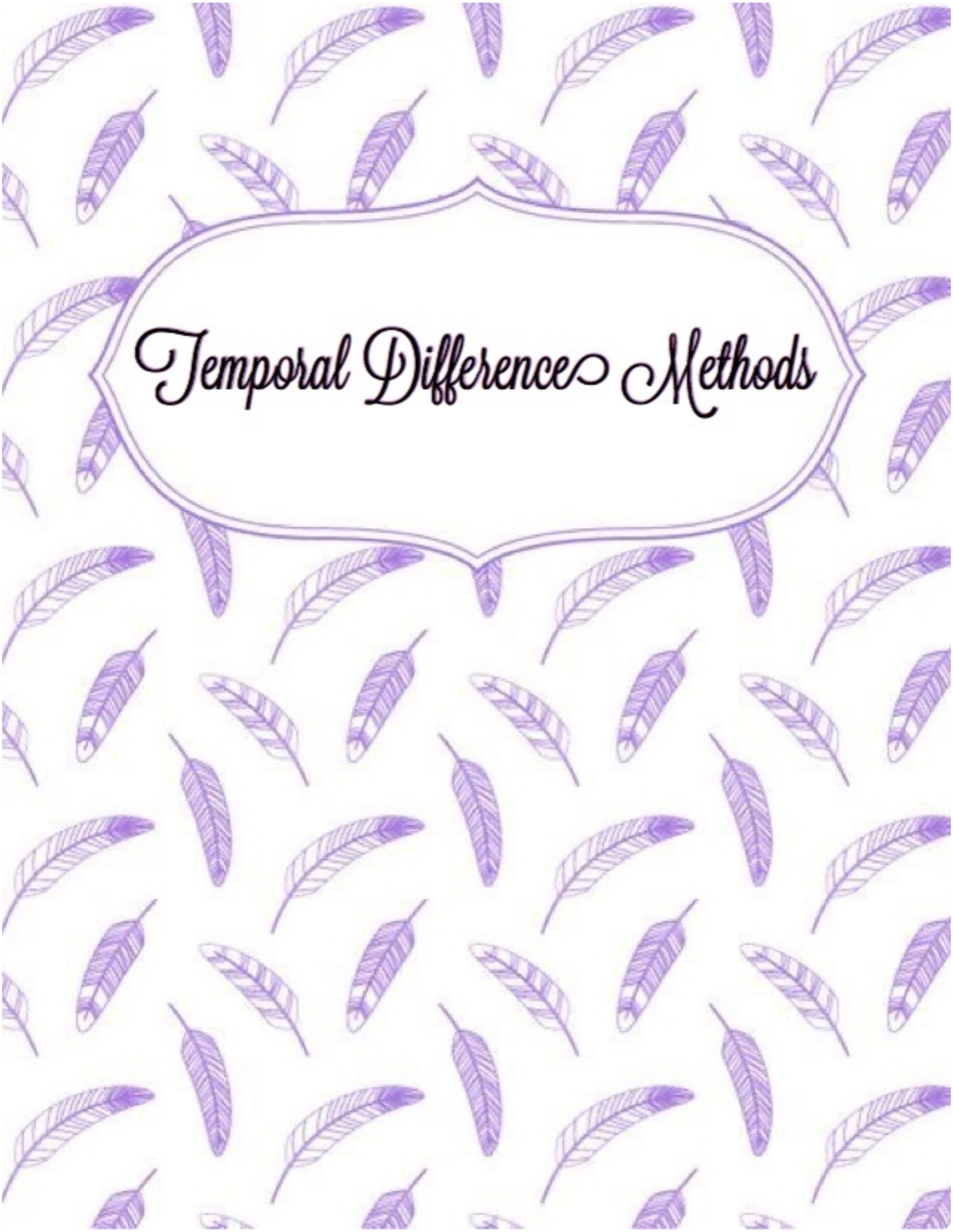
Temporal Difference Methods

Monte Carlo methods require to complete an episode of interaction before updating the Q-Table. Temporal Difference methods will instead update the Q-Table after every time-step.

SARSA :-

S A R S A

State - Action - Reward - Next State - Next Action

From Monte Carlo (MC) Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (G_t - Q(S_t, A_t))$$

Algorithm 13: Sarsa

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\text{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ **to** *num_episodes* **do**
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  Choose action A_0 using policy derived from Q (e.g., ϵ-greedy)
  $t \leftarrow 0$
  repeat
    Take action A_t and observe R_{t+1}, S_{t+1}
    Choose action A_{t+1} using policy derived from Q (e.g., ϵ-greedy)
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))$
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q

From Temporal Difference (TD) Control

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (\underbrace{R_{t+1}}_{\text{Reward by } S_t, A_t} + \underbrace{\gamma Q(S_{t+1}, A_{t+1})}_{\text{Discounted return of next state, action where action is choosen using } \epsilon \text{ greedy policy}} - Q(S_t, A_t))$$

Q-Learning (Sarsamax) :-

In Q-learning, the next state's action is choosen using greedy policy instead of ϵ-greedy policy to always choose the optimally maximizing action.

i.e.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (R_{t+1} + \gamma \underbrace{\max_{a \in A} Q(S_{t+1}, a)}_{\text{choose action 'a' which gives the maximum return on state } S_{t+1}} - Q(S_t, A_t))$$

Expected Sarsa :-

In this we take the sum of the expected return from each (s, a) pair i.e. we take the probability of that action happening into account.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (R_{t+1} + \gamma \underbrace{\sum_{a \in A} \pi(a|S_{t+1}) \cdot Q(S_{t+1}, a)}_{\text{Prob. to choose action 'a' given state } S_{t+1} \text{ in the } \epsilon\text{-greedy policy.}} - Q(S_t, A_t))$$

For eg. if $\epsilon = 0.4$ & action set are $\Rightarrow S$ $\overset{a_1}{[+8}$ $\overset{a_2}{+7}$ $\overset{a_3}{+9}$ $\overset{a_4}{+8]}$

then prob. of each action to be choosen based on ϵ-greedy policy will be :-

a_3 = maximum (+9), hence it will be choosen with $(1-\epsilon) = 0.6$ prob.

For all the actions, there is a 0.4 prob. to be choosen. i.e.

$0.4/4 = 0.1$

$$\pi(a|s) = \begin{matrix} a_1 & a_2 & a_3 & a_4 \\ [\ 0.1 & 0.1 & 0.7 & 0.1\ ] \end{matrix}$$

$0.6 + 0.1$

Optimism :- we always initialize the Q-Table. It has been shown that initializing the estimates to large values can improve performance.

For eg. if all the rewards that can be received are negative, the initialize Q-Table with zero.

Differences :-

- Sarsa & Expected Sarsa are both on-policy TD control algorithms. In this case, the same (ϵ-greedy) policy that is evaluated & improved is also used to select actions.
- Sarsamax (Q-learning) is an off-policy method, where the (ϵ-greedy) policy that is evaluated & improved is different from the (greedy) policy used to select actions.
- On-policy have better online performance than Off-policy methods.
- Expected-Sarsa achives better performance than Sarsa.

RL Algorithms

Model Based Learning (Dynamic Programming)

- Policy Iteration
- Value Iteration
- Requires a model of state transitions & rewards
- Uses DP to iteratively compute value & policy using the model

Model Free Learning

- Monte Carlo
- Temporal Difference
- Don't require a explicit model
- Sample the model by doing exploratory actions & use the experience gained to estimate the value functions.