

# Lecture notes - Introduction to Reinforcement learning with David Silver

## Lecture 5

By - Amal Sunny

# Model-Free control

---

- So this lecture deals with when we drop an agent into an unknown environment- what actions should it take to maximize reward
- Control - was to discover the optimal value function for a given MDP

## On-policy and Off-policy

- On-policy learning
  - Learns on the job - i.e while following policy
  - Learns about policy  $\pi$  from experience sampled from said policy  $\pi$
- Off-policy learning
  - Learning from someone else's experience
  - Learning about policy  $\pi$  from experience sampled from  $\mu$ , a different policy.

## On-policy Monte-Carlo Control

---

### Generalized policy iteration

- Recaping what was the generalize policy iteration process.

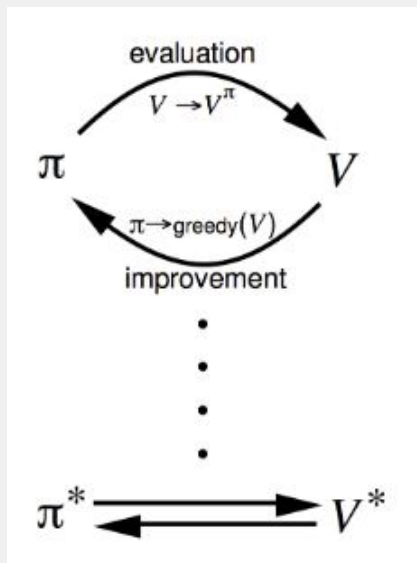
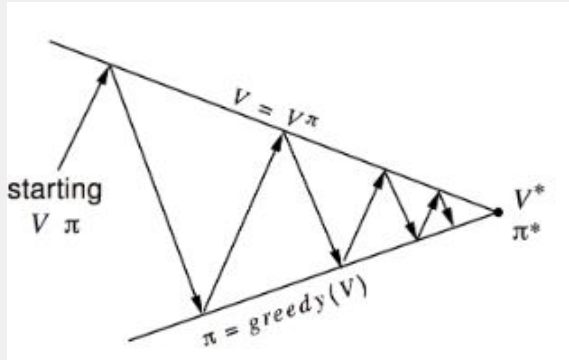
- To find best policy - given a certain policy, how do we improve on it ?
- Given a policy  $\pi$ 
  - We first evaluate the policy  $\pi$  - we evaluate expected reward under it.

- $$v_{\pi}(s) = E[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$$

- Improve the policy by acting greedily w.r.t to the data we just obtained from  $v_{\pi}$

- $$\pi' = greedy(v_{\pi})$$

- We iterate over these two processes enough times.
- Eventually, this process **always** converges to optimal policy.



## Generalized policy iteration with MC-evaluation

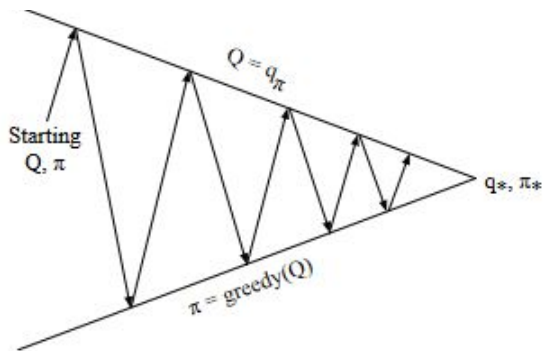
- For MC evaluation, we're gonna try to substitute the policy as MC and evaluate under it, while still retaining the greedy policy improvement part
- Essentially,
  - Policy evaluation: Monte-Carlo policy evaluation,  $V = v_\pi$  ?
  - Policy improvement: Greedy policy improvement ?
- However, there are issues with this approach. Firstly with the evaluation assumption:
  - The algorithm is supposed to be model free, but for greedy policy improvement, we need the model to update the equation (cuz we don't know  $P_{ss'}^a$ )

$$\pi'(s) = \operatorname{argmax}_{a \in A} R_a^s + P_{ss'}^a V(s')$$

## Generalised Policy Iteration with Action-Value Function

- To overcome that, we implement greedy policy improvement over  $Q(s,a)$  - it is model free.

$$\pi'(s) = \operatorname{argmax}_{a \in A} Q(s, a)$$



- So now we have,
  - Policy evaluation: Monte-Carlo policy evaluation,  $Q = q_0$
  - Policy improvement: Greedy policy improvement ?
- Another problem arises from the improvement part, if we're improving it greedily we can get stuck without converging to maximum. The right states we need to evaluate might go undiscovered under greedy exploration.

## Exploration

- Under that, we come to exploration - we can't decide if a certain policy is optimum if it does not explore all states. We might get stuck at some local optimum and not be able to find the global optimum unless all states are explored.

### $\epsilon$ – Greedy Exploration

- Is a very simple approach to ensuring continual exploration
- All  $m$  actions are tried with non-zero probability
- So essentially, we try a random action with probability  $\epsilon$
- And otherwise, the greedy action is followed with probability  $1 - \epsilon$

$$\pi(a|s) = \begin{cases} \epsilon/m + 1-\epsilon & \text{if } a^* = \operatorname{argmax}_{a \in A} Q(s, a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

- This policy might sound naive, but it ensures exploration and improvement of policy

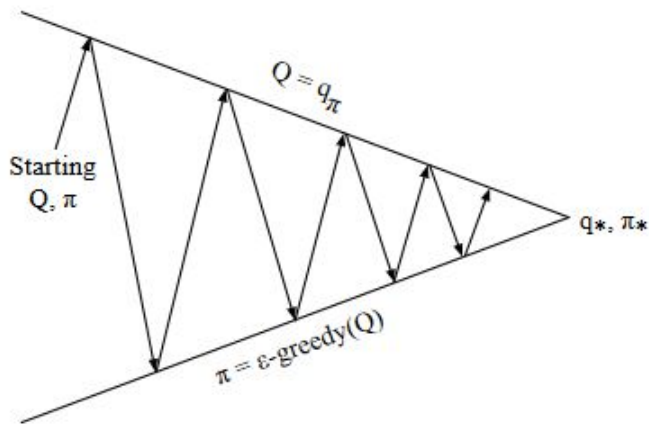
#### Theorem

For any  $\epsilon$ -greedy policy  $\pi$ , the  $\epsilon$ -greedy policy  $\pi'$  with respect to  $q_\pi$  is an improvement,  $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} q_\pi(s, \pi'(s)) &= \sum_{a \in A} \pi'(a|s) q_\pi(s, a) \\ &= \epsilon/m \sum_{a \in A} q_\pi(s, a) + (1 - \epsilon) \max_{a \in A} q_\pi(s, a) \end{aligned}$$

$$\begin{aligned} &\geq \epsilon / m \sum_{a \in A} q_{\pi}(s, a) + (1 - \epsilon) \sum_{a \in A} \pi(a|s) - \epsilon / m 1 - \epsilon q_{\pi}(s, a) \\ &= \sum_{a \in A} \pi(a|s) q_{\pi}(s, a) = v_{\pi}(s) \end{aligned}$$

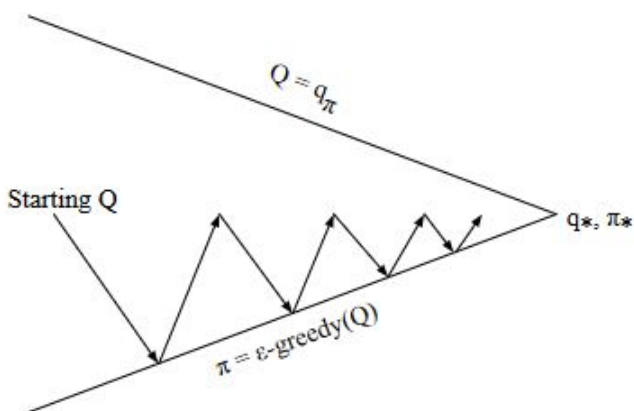
- Now, we have our MC policy iteration as
  - Policy evaluation: Monte-Carlo policy evaluation,  $Q = q_{\pi}$
  - Policy improvement:  $\epsilon$ -greedy policy improvement



- However, this process of evaluation might still take too long to realistically iterate through all states.

## Monte-Carlo Control

- Instead of waiting till we generate a whole batch of episodes, we update the policy evaluation based on end of each episode.
- This way we change the rate of improvement of the policy and converge faster



- Every episode
  - Policy evaluation: Monte-Carlo policy evaluation,  $Q \approx q_{\pi}$
  - Policy improvement:  $\epsilon$ -greedy policy improvement

## GLIE (Greedy in limit of infinite exploration)

### Definition

- All state-action pairs are explored infinitely many times,

$$\lim_{k \rightarrow \infty} N_k(s, a) = \infty$$

- The policy converges on a greedy policy,

$$\lim_{k \rightarrow \infty} \pi_k(a|s) = 1(a = \operatorname{argmax}_{a' \in A} Q_k(s, a'))$$

- One way to achieve this with  $\epsilon$ -greedy, is if we set  $\epsilon_k = 1/k$  - it will decay to 0 when  $k \rightarrow \infty$

## GLIE Monte-Carlo Control

- We sample the  $k^{th}$  episode using our existing policy  $\pi$ :  $\{S_1, A_1, R_2, \dots, S_T\} \sim \pi$
- Policy evaluation
  - For each state  $S_t$  and action  $A_t$  in the episode - we update the action-value function incrementally at the end of each episode.

$$\begin{aligned} N(S_t, A_t) &\leftarrow N(S_t, A_t) + 1 \\ Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t)) \end{aligned}$$

- Policy improvement
  - The policy is improved based on new action-value function under GLIE.

$$\begin{aligned} \epsilon &\leftarrow 1/k \\ \pi &\leftarrow \epsilon\text{-greedy}(Q) \end{aligned}$$

### Theorem

GLIE Monte-Carlo control converges to the optimal action-value function,  $Q(s, a) \rightarrow q_*(s, a)$

- The initial value of Q for this algorithm does not matter (for most algos it doesn't - apart from for performance reasons, but here not even that)

## On-policy Temporal-Difference learning

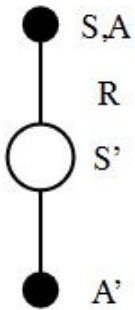
### MC vs TD control

- TD has advantages over MC, namely (refer week 4):
  - Lower variance
  - Online
  - Can be used on incomplete sequences

- Natural idea to apply TD here: use TD instead of MC in our evaluation loop
  - Apply TD to  $Q(S,A)$
  - Use  $\epsilon$ -greedy policy improvement
  - Update every time-step(online)

## SARSA

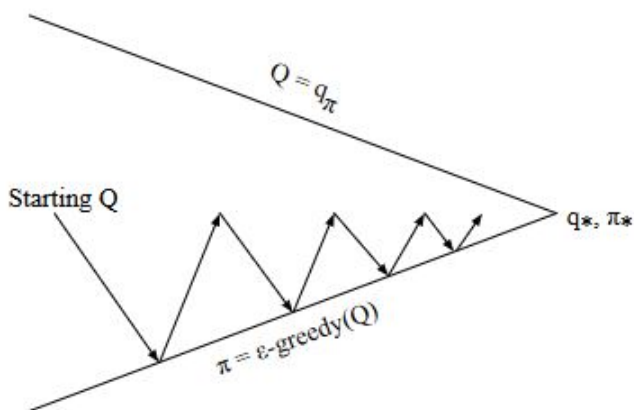
- The algorithm we get from applying TD learning on policy under  $\epsilon$ -greedy is SARSA.
- The name comes from this digramatic explanation of it below:



- We start of in a state  $S$ , take an action  $A$  under our policy and sample the reward  $R$  from the environment.
- We end up in the state  $S'$  and we apply our policy again to figure out the next action  $A'$ (ergo  $S,A,R,S',A$ )
- These values are plugged into the bellman equation to update our action-value function:
  - $R + \gamma Q(S', A')$  is our TD target to update towards

$$Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$$

## On-policy control with SARSA



So for on-policy control with SARSA

- Every time-step:
  - Policy evaluation: Sarsa,  $Q \approx q_\pi$

- Policy improvement:  $\epsilon$ -greedy policy improvement

## SARSA algorithm for On-policy control

```

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
  Initialize  $S$ 
  Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
  Repeat (for each step of episode):
    Take action  $A$ , observe  $R, S'$ 
    Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$ 
     $S \leftarrow S'; A \leftarrow A';$ 
  until  $S$  is terminal

```

## Convergence of SARSA

### Theorem

Sarsa converges to the optimal action-value function,  $Q(s, a) \rightarrow q_*(s, a)$ , under the following conditions:

- GLIE sequence of policies  $\pi_t(a|s)$  - so that everything is explored and the policy becomes greedy at the end
- Robbins-Monro sequence of step-sizes  $\alpha_t$  - which are:
  - $\sum_{t=1}^{\infty} \alpha_t = \infty$  - which means the Q value can be moved very far (like very far from your initial estimate)
  - $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$  - means changes to the Q value diminish eventually (become exponentially smaller)
- Note: Usually SARSA works even without these two points, but that's an empirical result rather than a theoretical one.
- While training SARSA for a problem, the first episode takes a lot of time steps- but subsequent episodes take much less steps

## n-step SARSA

- Considering the n-step version of TD from before, surely we can try it here to get best of both worlds (more steps and online learning)
- So, we have n-step returns for  $n = 1, 2, \dots, \infty$ :

$$\begin{array}{ll}
 n = 1 & (\text{Sarsa}) \quad q_t^{(1)} = R_{t+1} + \gamma Q(S_{t+1}) \\
 n = 2 & q_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}) \\
 & \vdots \\
 n = \infty & (\text{MC}) \quad q_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T
 \end{array}$$

- Thus we have n-step Q-return as:

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n})$$

- And insert this as our target for SARSA updates, in the equation:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(q_t^{(n)} - Q(S_t, A_t))$$

## Forward View SARSA( $\lambda$ )

- Just like in TD( $\lambda$ ) forward view, we create  $q^\lambda$  return that combines all n-step Q-returns  $q_t^{(n)}$
- Using weight  $(1 - \lambda)\lambda^{n-1}$

$$q_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} q_t^{(n)}$$

- Plugging that into our equation:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(q_t^\lambda - Q(S_t, A_t))$$

## Backward View SARSA( $\lambda$ )

- Just like in TD( $\lambda$ ) backward view, we use eligibility traces in an online algorithm.
- SARSA( $\lambda$ ) has one eligibility trace for each state-action pair

$$E_0(s, a) = 0$$

$$E_t(s, a) = \gamma \lambda E_{t-1}(s, a) + 1(S_t = s, A_t = a)$$

- Every time we visit that state, its value is increased - otherwise it decays for every time step we don't visit it.
- $Q(s, a)$  is updated for every state  $s$  and action  $a$  - in proportion to TD-error  $\delta_t$  and eligibility trace  $E_t(s, a)$

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha \delta_t E_t(s, a)$$

## Algorithm

```

Initialize  $Q(s, a)$  arbitrarily, for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ 
Repeat (for each episode):
     $E(s, a) = 0$ , for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ 
    Initialize  $S, A$ 
    Repeat (for each step of episode):
        Take action  $A$ , observe  $R, S'$ 
        Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
         $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$ 
         $E(S, A) \leftarrow E(S, A) + 1$ 
        For all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ :
             $Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$ 
             $E(s, a) \leftarrow \gamma \lambda E(s, a)$ 
         $S \leftarrow S'; A \leftarrow A'$ 
    until  $S$  is terminal

```



---

# Off-policy learning

---

- So, we evaluate **one** policy - our target policy  $\pi(a|s)$  to compute  $v_\pi(s)$  or  $q_\pi(s, a)$
- While following an entirely different behaviour policy (i.e actions we take)  $\mu(a|s)$

$$S_1, A_1, R_2, \dots, S_T \sim \mu$$

- This can be important for:
  - Learning from other agents/observing humans
  - Re-use experience generated from older policies
  - Learn about optimal policy while following exploratory policy
  - Learn about multiple policies while following one policy

## Importance Sampling

- We estimate the expectation of one distribution by means of another without actually calculating for both of them independently.
- So we plan to use this to infer other policies from just calculating them for one policy

$$\begin{aligned} E_{X \sim P}[f(X)] &= \sum P(X) f(X) \\ &= \sum Q(X) \frac{P(X)}{Q(X)} f(X) \\ &= E_{X \sim Q} \left[ \frac{P(X)}{Q(X)} f(X) \right] \end{aligned}$$

## Importance Sampling for Off-Policy Monte-Carlo

- Use returns generated from  $\mu$  to evaluate  $\pi$  - using importance sampling
- Weight return  $G_t$  according to similarity b/w policies
- And then take the ratio for **all** the steps according to importance sampling to get a corrected return
- This return is then plugged in as the target for the bellman equation to update value function
- However, this sampling for MC is near useless due to the high variance introduced by this method (combined with MC's inherent variance that makes it useless)

## Importance Sampling for Off-Policy TD

- Thus, we turn to TD learning to actually use this
- For TD, we just important sample over one step instead of all of them
- The TD target just gets divided by the ratio needed for Importance sampling
- This method is much lower variance than Monte-Carlo importance sampling

- Policies only need to be similar over a single step

## Q-learning

- Idea works best with off-policy learning
- Specific to TD(0)
- No importance sampling is required
- Under this, our next action is chosen using behaviour policy  $A_{t+1} \sim \mu(\cdot | S_t)$
- But we take in account the alternative policy successor action  $A' \sim \pi(\cdot | S_t)$
- And update  $Q(S_t, A_t)$  towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t))$$

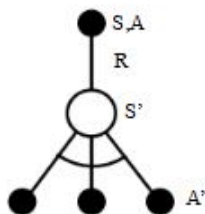
## Off-Policy Control with Q-Learning

- Well-known version of Q-learning
- Special case where:
  - The target policy  $\pi$  is greedy w.r.t.  $Q(s, a)$
- The Q-learning target then simplifies:

$$\pi(S_{t+1}) = \operatorname{argmax}_{a'} Q(S_{t+1}, a')$$

- The behaviour policy  $\mu$  is e.g.  $\epsilon$ -greedy w.r.t.  $Q(s, a)$

$$R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')$$



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

Theorem

Q-learning control converges to the optimal action-value function,  $Q(s, a) \rightarrow q_*(s, a)$

### Algorithm

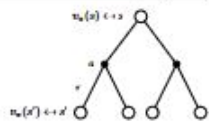

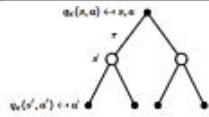
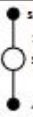
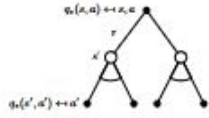
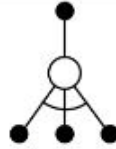
Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$   
 Repeat (for each episode):

```

Initialize  $S$ 
Repeat (for each step of episode):
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
    Take action  $A$ , observe  $R, S'$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
     $S \leftarrow S'$ ;
until  $S$  is terminal

```

## Relationship Between DP and TD

|  | Full Backup (DP)   | Sample Backup (TD)   |
|--|--|--|
| Bellman Expectation Equation for $v_{\pi}(s)$    |  <p>Iterative Policy Evaluation</p> |  <p>TD Learning</p> |
| Bellman Expectation Equation for $q_{\pi}(s, a)$ |  <p>Q-Policy Iteration</p>          |  <p>Sarsa</p>       |
| Bellman Optimality Equation for $q_{*}(s, a)$    |  <p>Q-Value Iteration</p>          |  <p>Q-Learning</p> |

| Full Backup (DP)   | Sample Backup (TD)   |
|--|--|
| Iterative Policy Evaluation  | TD Learning  |
| $V(s) \leftarrow \mathbb{E}[R + \gamma V(S') \mid s]$  | $V(S) \xleftarrow{\alpha} R + \gamma V(S')$                                  |
| Q-Policy Iteration   | Sarsa  |
| $Q(s, a) \leftarrow \mathbb{E}[R + \gamma Q(S', A') \mid s, a]$                                      | $Q(S, A) \xleftarrow{\alpha} R + \gamma Q(S', A')$                           |
| Q-Value Iteration  | Q-Learning   |
| $Q(s, a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') \mid s, a\right]$ | $Q(S, A) \xleftarrow{\alpha} R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')$ |

**Problem 1** - Taken from CS234

### 1) [CA Session] To Converge Or Not To Converge

Recall the TD-learning update:  $V(s_t) = V(s_t) + \alpha([r_t + \gamma \cdot V(s_{t+1})] - V(s_t))$  and the Incremental Monte-Carlo update:  $V(s_t) = V(s_t) + \alpha(G_t - V(s_t))$ .

The convergence of these value functions is highly dependent on the learning rate  $\alpha$ . In fact, it may be convenient to vary the learning rate based on the timestep. In this problem, we will explore how varying the learning rate may affect convergence.

Let us define our timestep-varying learning rate as  $\alpha_t$ . The above updates will converge with probability 1 if the following conditions are satisfied:

$$\sum_{i=1}^{\infty} \alpha_t = \infty \quad (1)$$

$$\sum_{i=1}^{\infty} \alpha_t^2 < \infty \quad (2)$$

Condition (1) is required to guarantee that the steps are large enough to eventually overcome any initial conditions or random fluctuations. Condition (2) guarantees that eventually the steps become small enough to assure convergence.

(a) Let  $\alpha_t = \frac{1}{n}$ . Does this  $\alpha_t$  guarantee convergence?

Ans:

While it was already taken as an example for valid learning rate step, we will prove it now.

For  $\sum_{i=1}^{\infty} \frac{1}{n} = \infty$ , since  $1/n$  is a harmonic series

For  $\sum_{i=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6} < \infty$  - which is the solution to the basel problem

As such it satisfied both of Robbin-Monroe criteria, and thus would be guaranteed to converge.

**Problem 2** - Cont.d from 1

b) Let  $\alpha_t = 1/2$ . Does this  $\alpha_t$  guarantee convergence?

Ans:

For first condition,

$$\sum_{i=1}^{\infty} \frac{1}{2} = \infty$$

So, it satisfies that condition

For second condition,

$$\sum_{i=1}^{\infty} \frac{1}{2^2} = \sum_{i=1}^{\infty} \frac{1}{4} = \infty \not< \infty$$

It violates the second condition. Thus it will not be an appropriate step size to guarantee convergence.

---

## Program Of The Week (PoTW)

- Design a program to successfully beat flappy bird(a game)