Thursday Learning Hour

Deep Q Learning

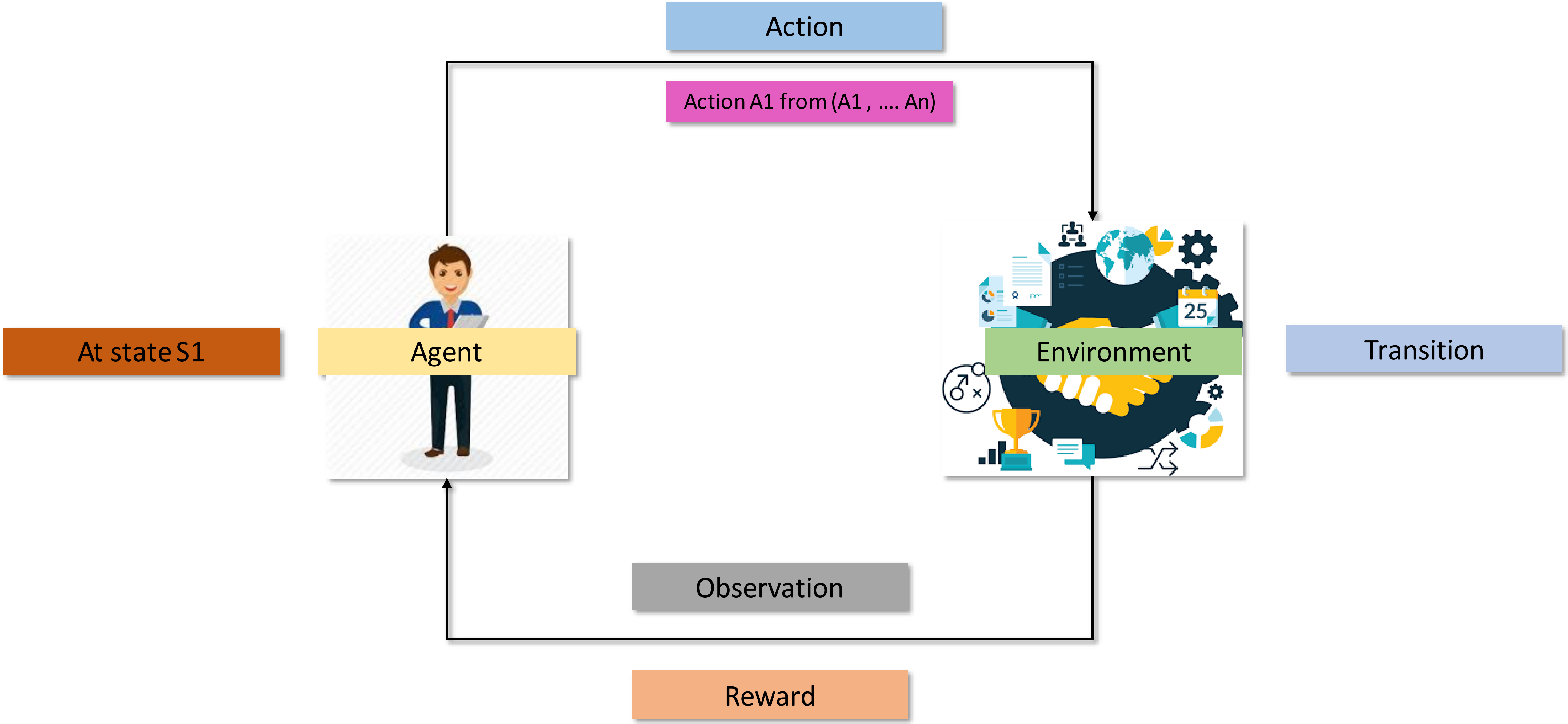
Prabakaran Chandran

25 March 2021

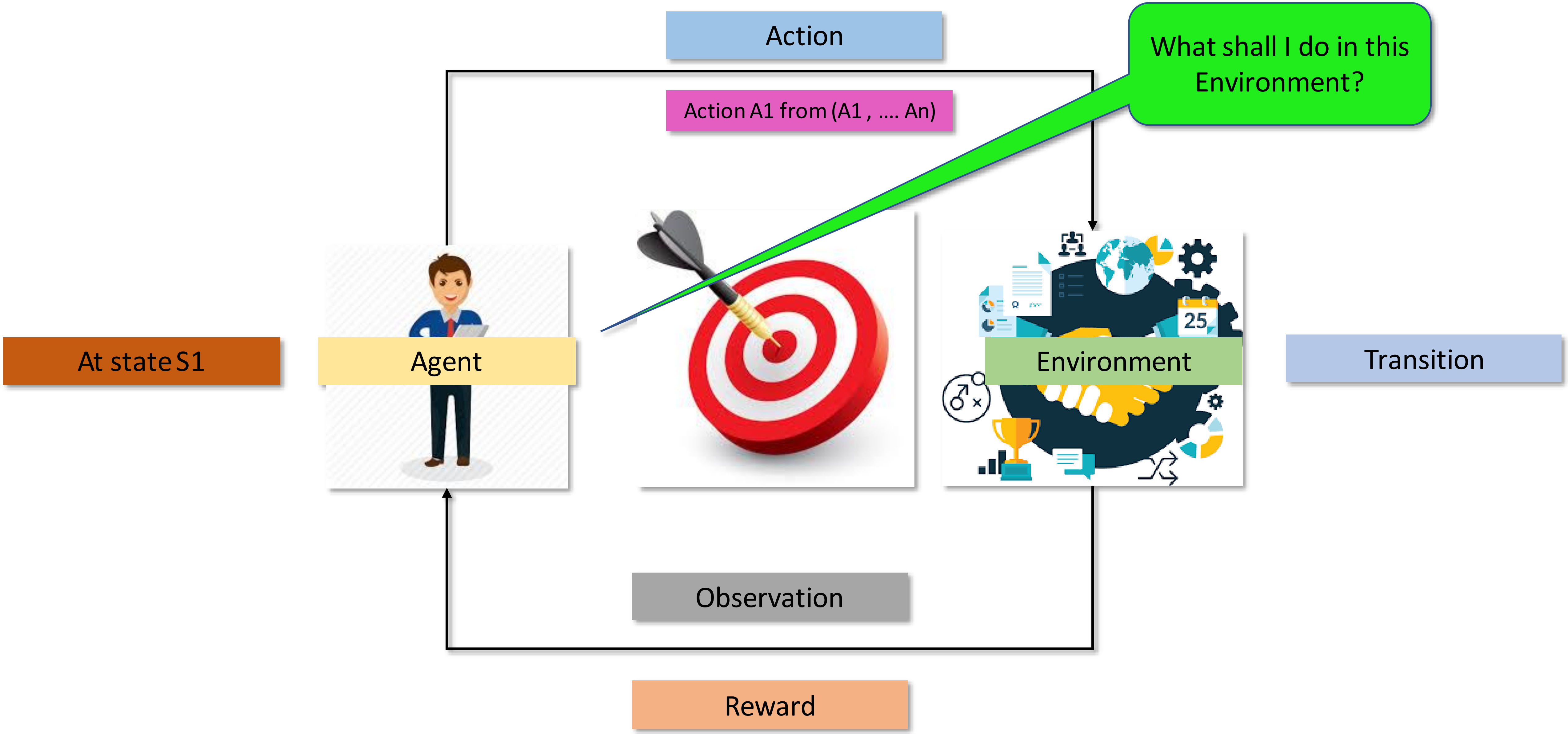
Agenda:

- Reinforcement Learning – A small Recap
- What is Q Function?
- Traditional Q Learning
- Deep Learning - overview
- Deep Q Learning - Architecture and Learning Algorithm
- Applications and Characteristics

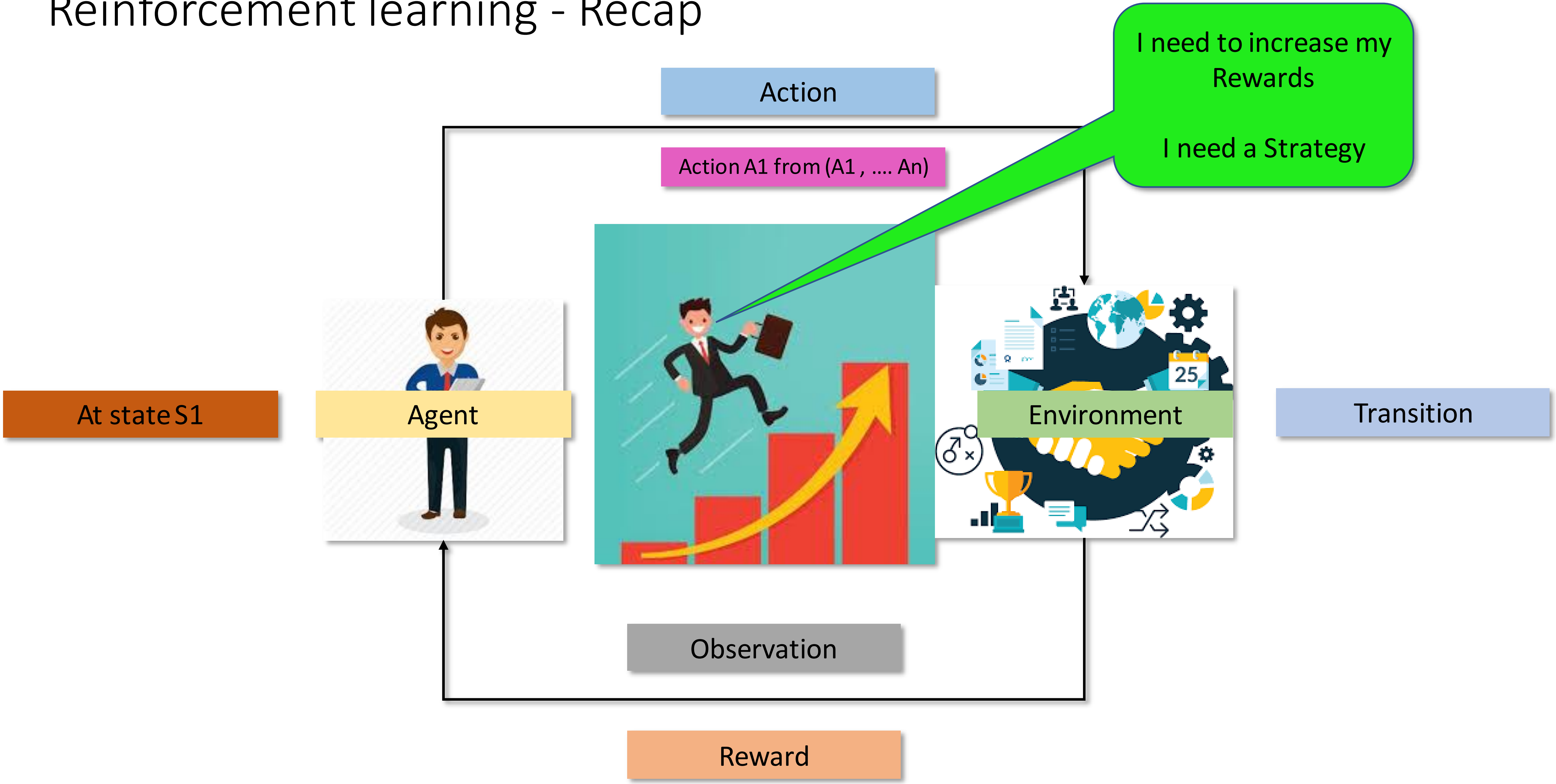
Reinforcement learning - Recap



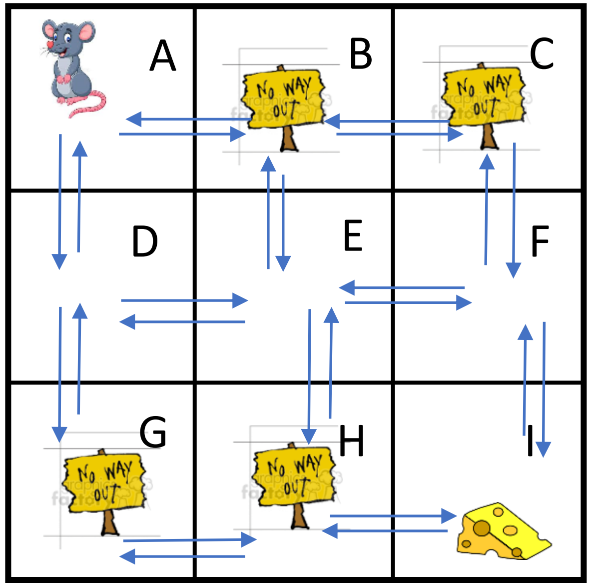
Reinforcement learning - Recap



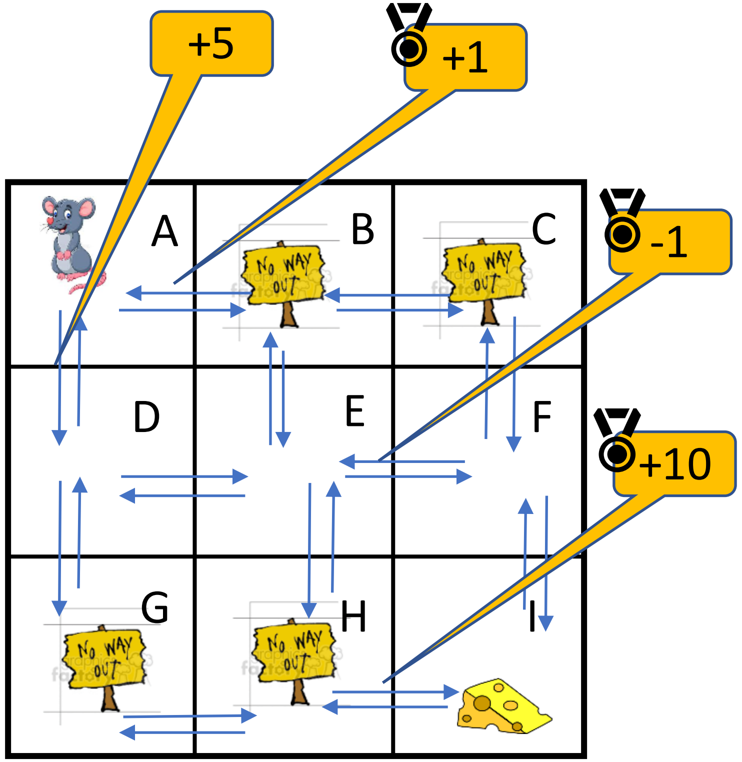
Reinforcement learning - Recap



Markov Reward Process:



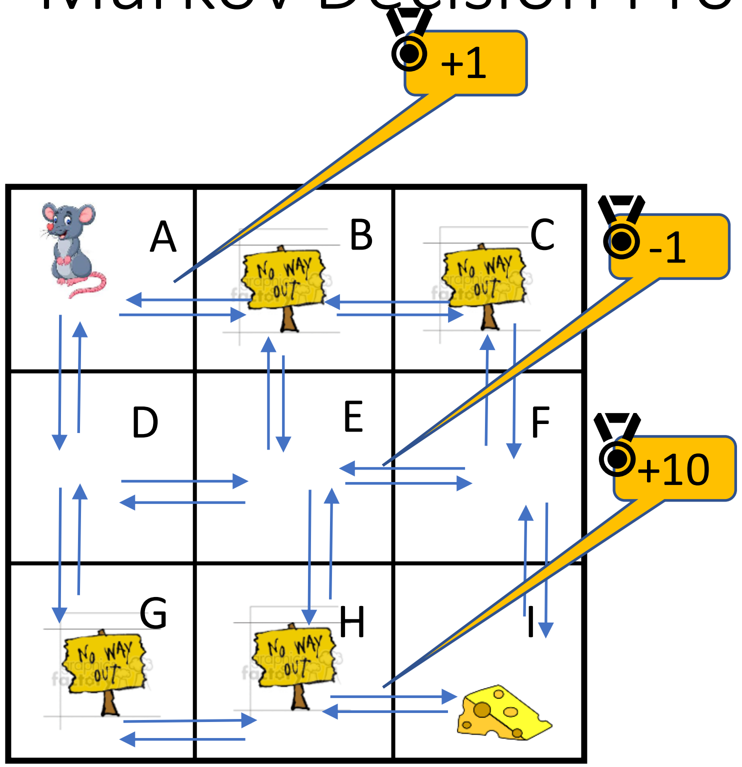
	A	B	C	D	E	F	G	H	I
A	0.2	0.1		0.7					
B	0.5	0.4	0.1						
C		0.4	0.2			0.4			
D	0.1			0.2	0.6		0.1		
E		0.1		0.3	0.2	0.3		0.1	
F			0.1		0.1	0.1			0.6
G					0.6		0.2	0.2	
H					0.1		0.1	0.1	0.7
I									



Possible State change (process / Chain)

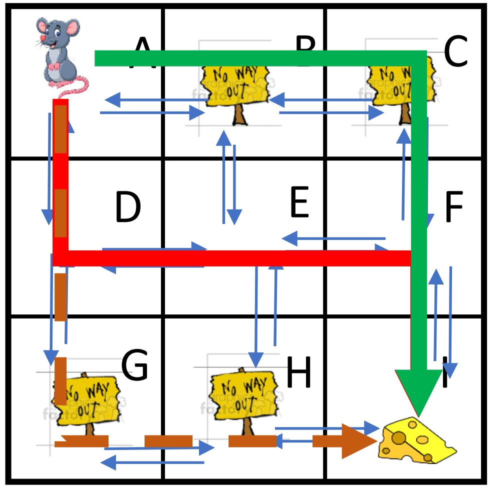
Each Transition will have its own reward point , Reward point can be positive or negative depends on their properties.
Each State can have a value , which is expected return by being in the state.
Ex: Value of A = 5 + 1 = 6(Expected Return)

Markov Decision Process:

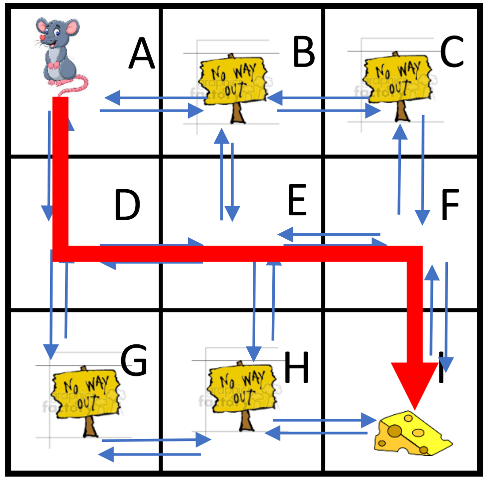


- At A (B , Right) and A(D, down) are the possible actions
- JD can select any one of the action (Decision he has to take)

He can take any Decision from the available action space , but there can be a negative reward too. This selection of action depends on Jeffery's Behavior (Policy)



Possible Decisions by selecting the Actions at each state.

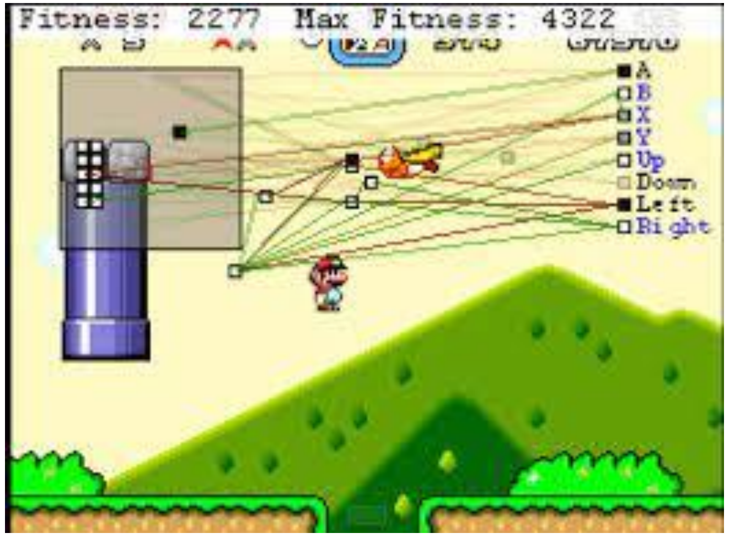


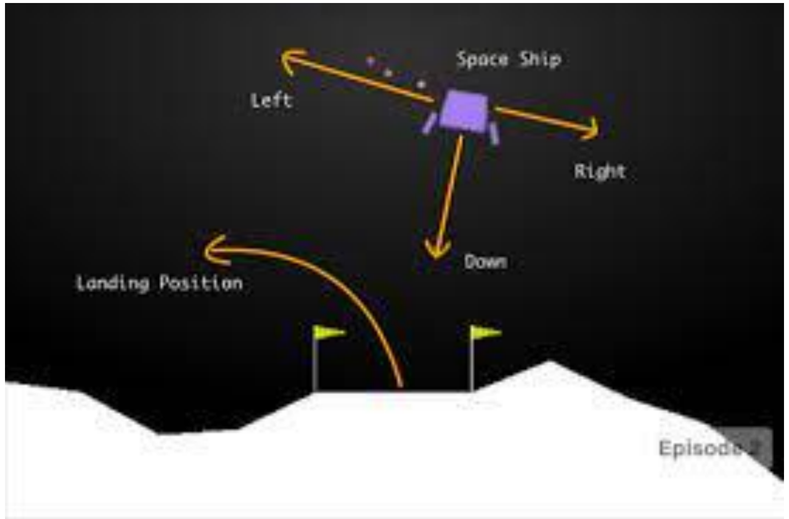
This is what we Jeffery need to follow to have more reward.

How can he learn this ??

By solving this MDP Enviornment

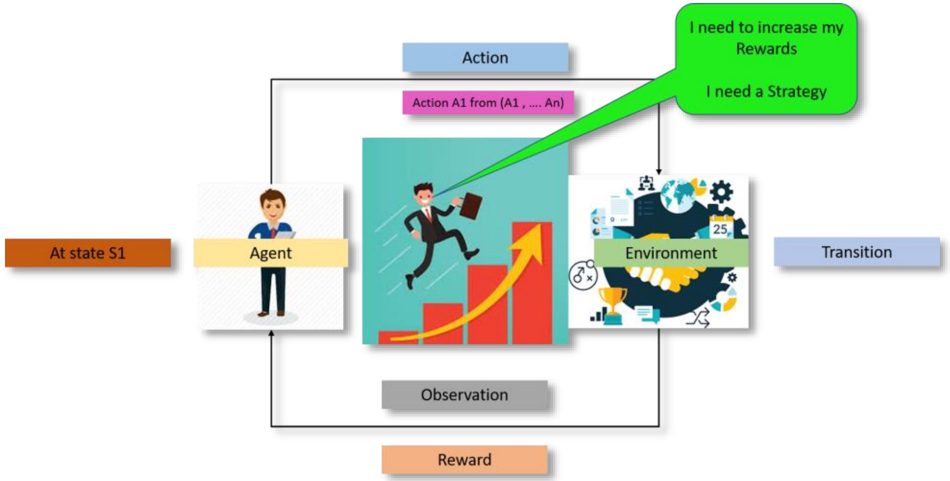
Few Intuitive Scenarios





How to Create that Strategy?

- In order to reach the Strategy, Solution is needed for the Environment.
- It means to reach the Optimal Policy, MDP has to be solved
- Optimal policy : $\pi* = \arg\mathit{max}\ E(R/\pi)$, the policy which gives more return
- How to reach optimal policy? → Solve the Environment



How to Solve the Environment / MDP ?

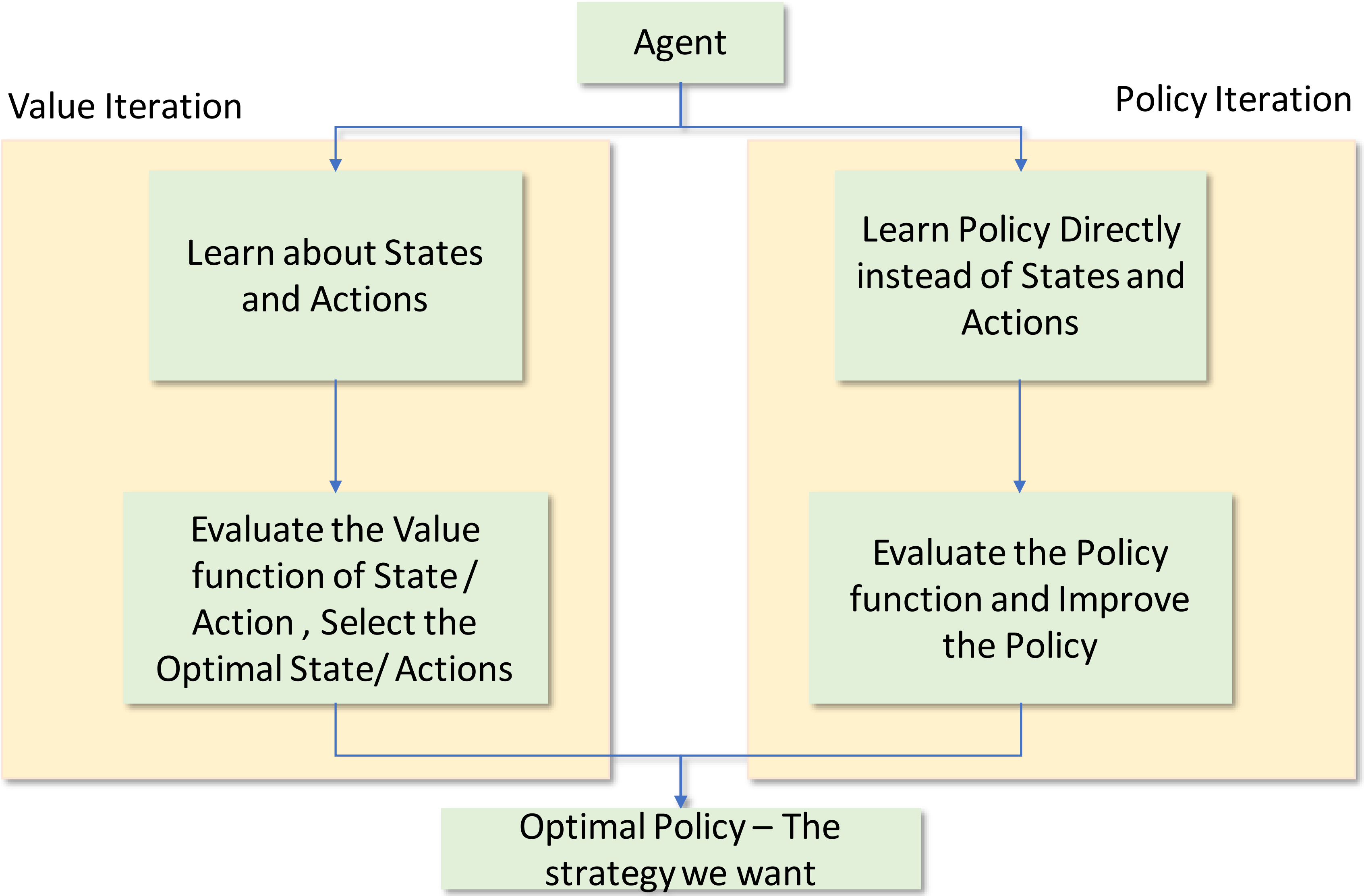
There are two ways:

1. *Go With the Cheat Sheet : Model-based RL uses experience to construct an internal model of the transitions and immediate outcomes in the environment. Appropriate actions are then chosen by searching or planning*

1. *Learn on the Go: Model-free RL, on the other hand, uses experience to learn directly one or both of two simpler quantities (state/ action values or policies) which can achieve the same optimal policy*

Model Free Learning:

In **Model-free** learning the agent relies on trial-and-error experience for setting up the optimal policy.



Learn through the Value Function:

$$V^{\pi}(s) = \mathbb{E}\big[R_t \quad | \quad s_t = s\big]$$

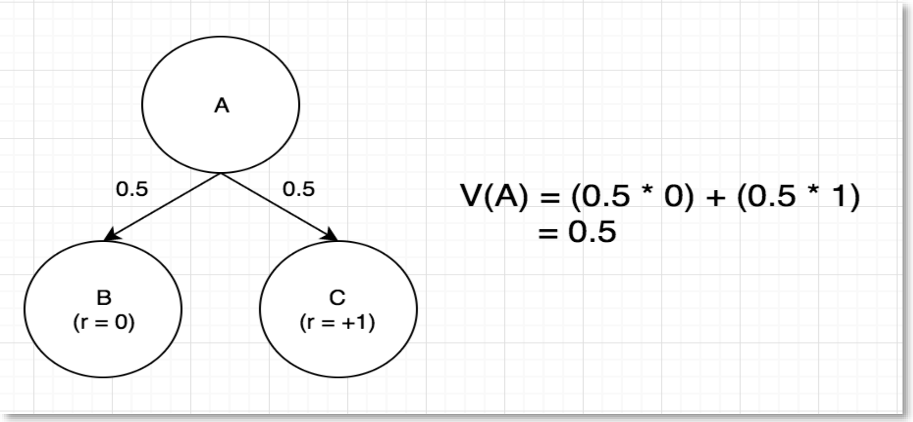
- Total Expected Return From the state / state – Action while following a policy
- To measure How good is a state to be in? / How good is an action to take?

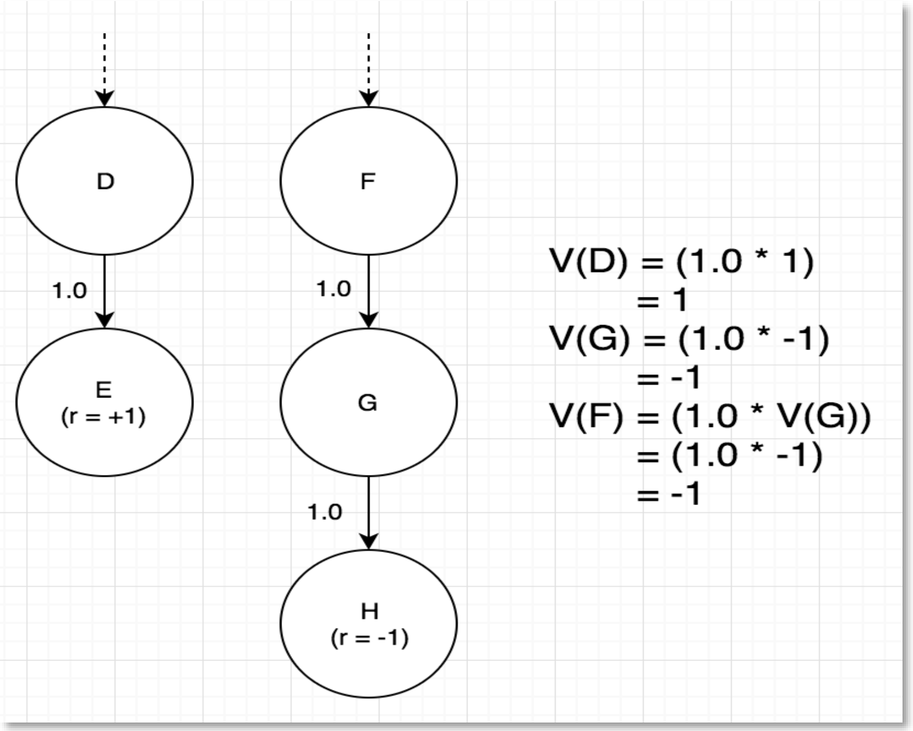
- Two Types of Value Functions are there – One is to evaluate State , another one is to evaluate an Action (Q Function)

- Action Value function defines the Quality of an action (A1) taken in a given state (s1)

$$Q^{\pi}(s,a) = \mathbb{E}\big[R_t \quad | \quad s_t = s, a_t = a\big]$$

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$





Learn through the Q Value Function:

$$Q^{\pi}(s,a) = \mathbb{E}\left[R_t \quad \middle| \quad s_t = s, a_t = a\right]$$

Bellman Equation

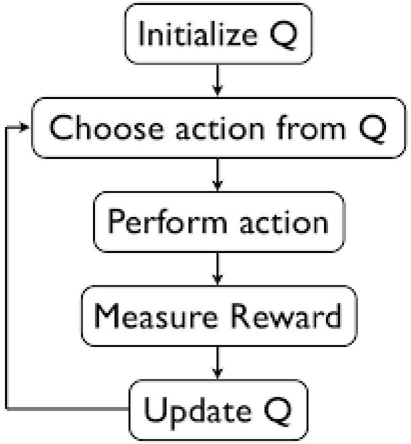
$$Q(s,a) = r + \gamma \max_{a'} Q(s', a')$$

- This Q Function help us to bring the Optimal State – Action pair because of its Greedy Behavior
- Quality here represents how useful a given action is in gaining some future reward.
- By Iterating the Q Value for all the State – Action pair until we reach the convergence in policy
- This sort of Q Value iteration is called as Q Learning
- For Every step Q Value will be updated based on the Evaluation
- If we have a Table of Q Value for each State – Action pair , An Agent Can use that as a Cheat Sheet.

Q Learning - Q Table iteration

- In Traditional Q Learning Method , Objective is to create a Exhaustive Q Table
- An Agent use the Q table as a Cheat sheet to Take state transition by selecting correct Action
- Temporal Difference Based Update Rule is used here.
- $Q_{t+1} = Q_t + \Delta Q \;\; ; New\ Q = Old\ Q + Temporal\ Difference$

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \bigg)}^{\text{temporal difference}}$$



- ➢ Q Learning Starts from Arbitrary Q Table
- ➢ Initial Action taken Randomly
- ➢ Reward will be measured for the current State action pair and Future State Action pairs
- ➢ Q Value will be update

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	327	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Deep Q Learning : Need of Deep Architecture

- Computation : Reinforcement learning can be sufficiently applicable to the environment where the all achievable states can be manged (iterated) and stored in standard computer RAM memory.

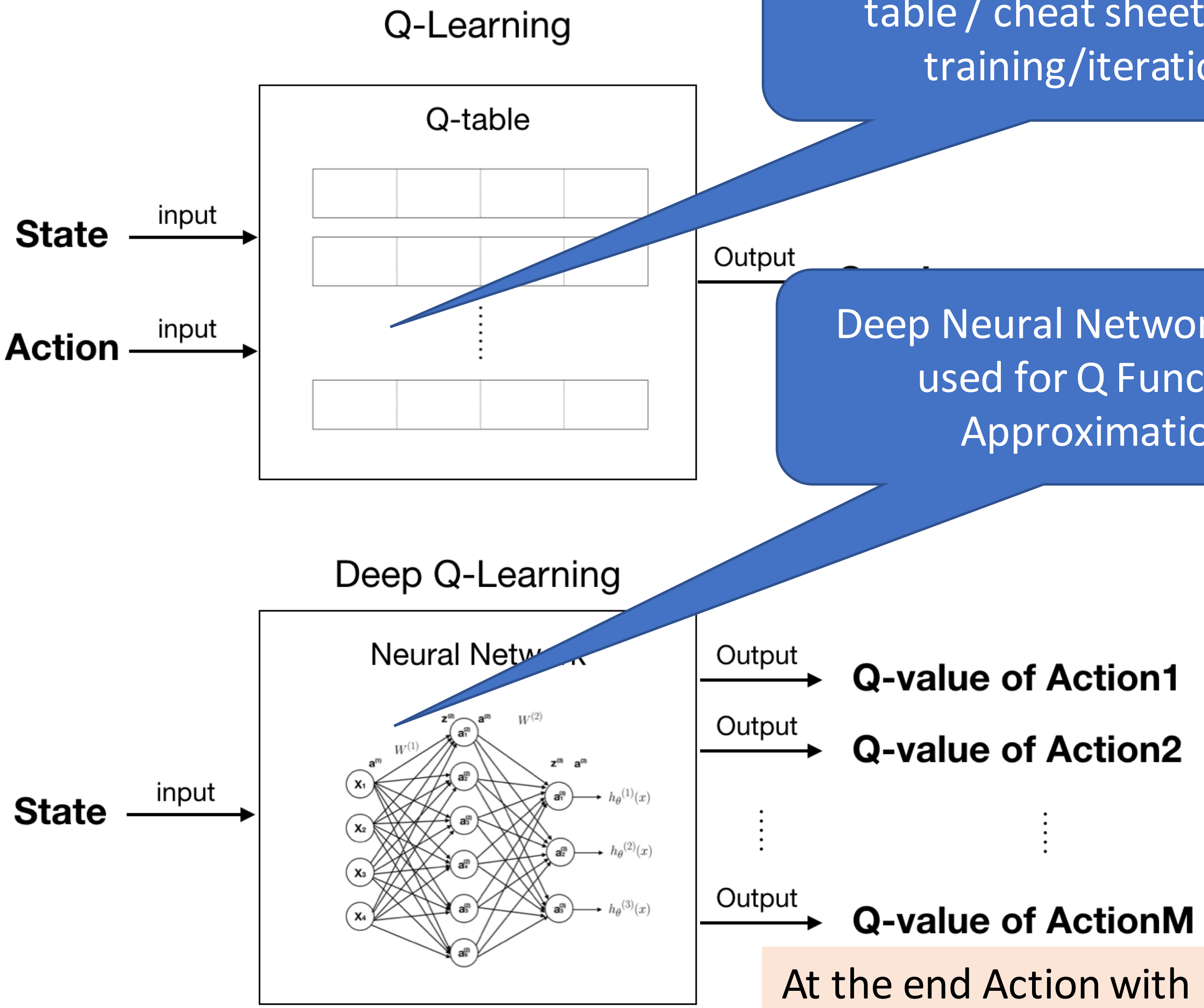
- Number of States : However, the environment where the number of states overwhelms the capacity of contemporary computers (for Atari games there are 12833600 states) the standard Reinforcement Learning approach is not very applicable.

- Furthermore, in real environment, the Agent has to face with continuous states (not discrete), continuous variables and continuous control (action) problems.

- Bearing in mind the complexity of environment the Agent has to operate in (number of states, continuous control) the standard well defined Reinforcement Learning Q — table is replaced by Deep Neural Network (Q — Network) which maps (non — linear approximation) environment states to Agent actions.

- Network architecture, choice of network hyper parameters and learning is performed during training phase (learning of Q — Network weight).

Deep Q Learning : Architecture



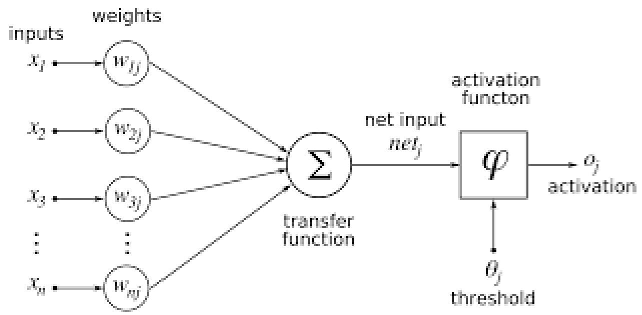
Q Table will be used as look up table / cheat sheet after training/iteration

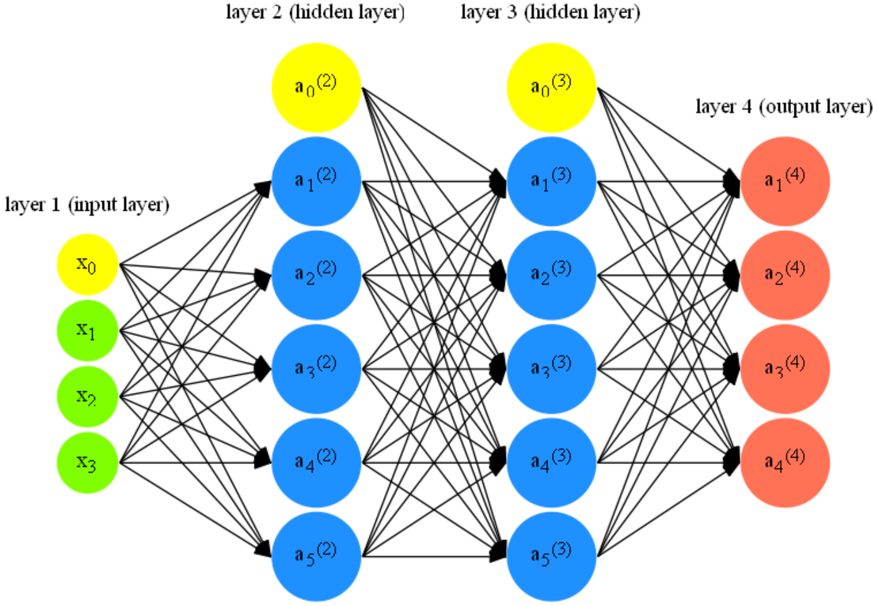
Deep Neural Network will be used for Q Function Approximation

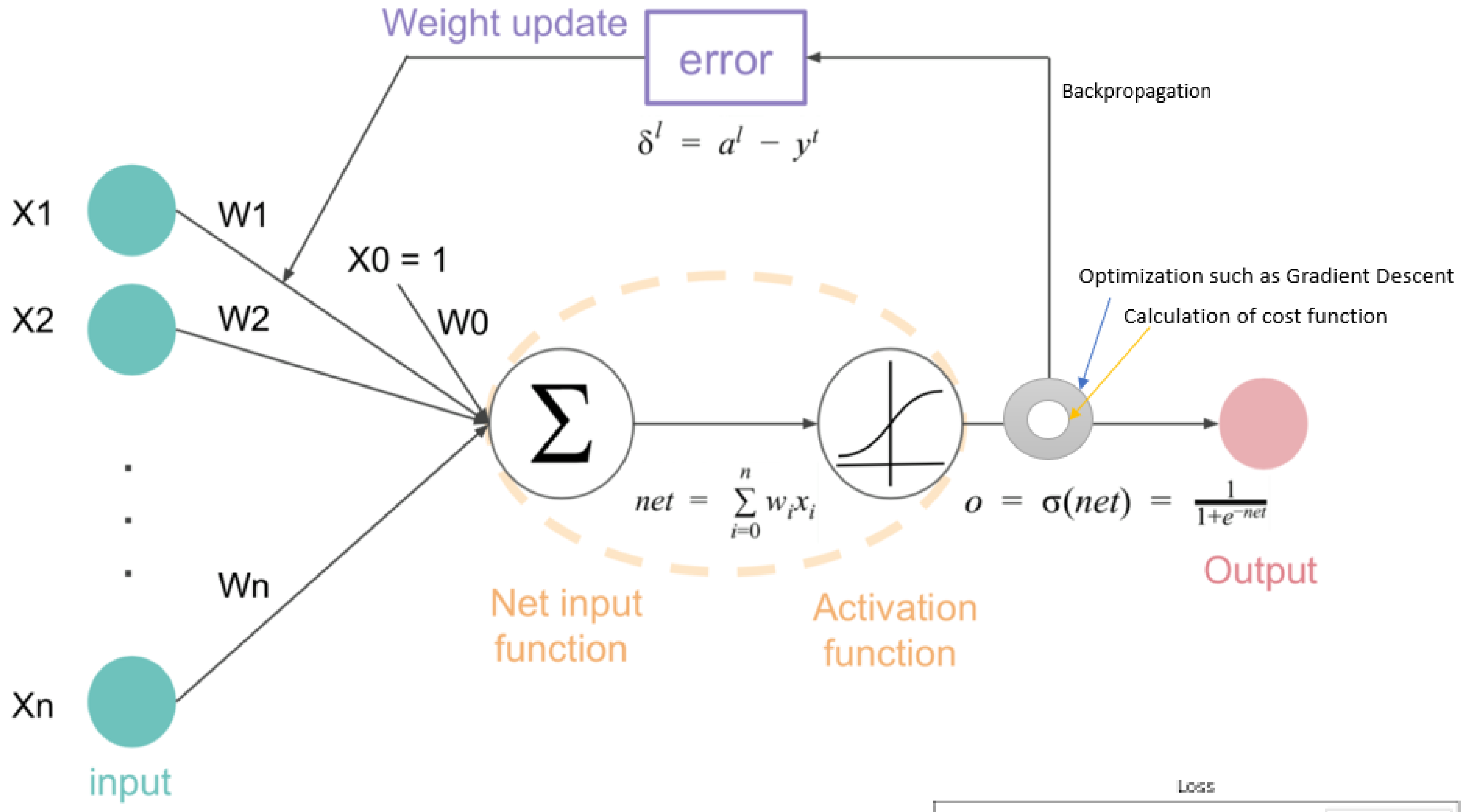
- Deep Q-Learning harness the power of deep learning with so-called Deep Q-Networks, or DQN for short.
- In this scenario, these networks are just standard feed forward neural networks which are utilized for predicting the best Q-Value.
- Advanced DQN networks use CNN layers to capture the states from Visual Environment (Games)
- In order for this approach to work, the agent has to store previous experiences in a local memory, but more on that later.

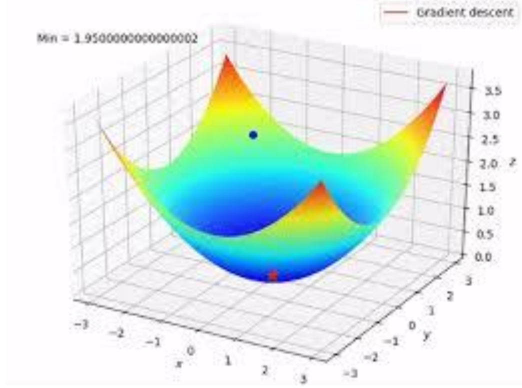
At the end Action with Max Q Value will be selected and Agents will move to next state. *best_action = arg max(DQN predicted Q-values).*

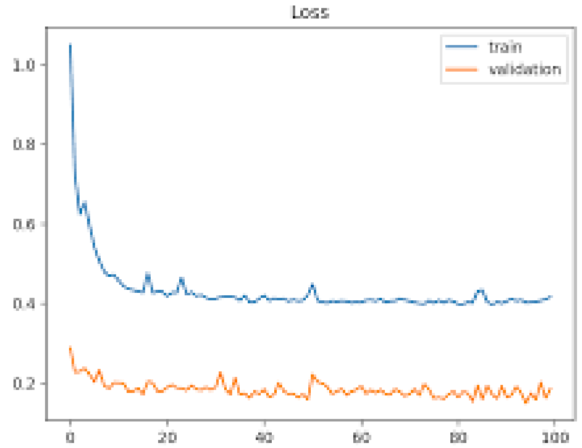
Deep Learning : overview



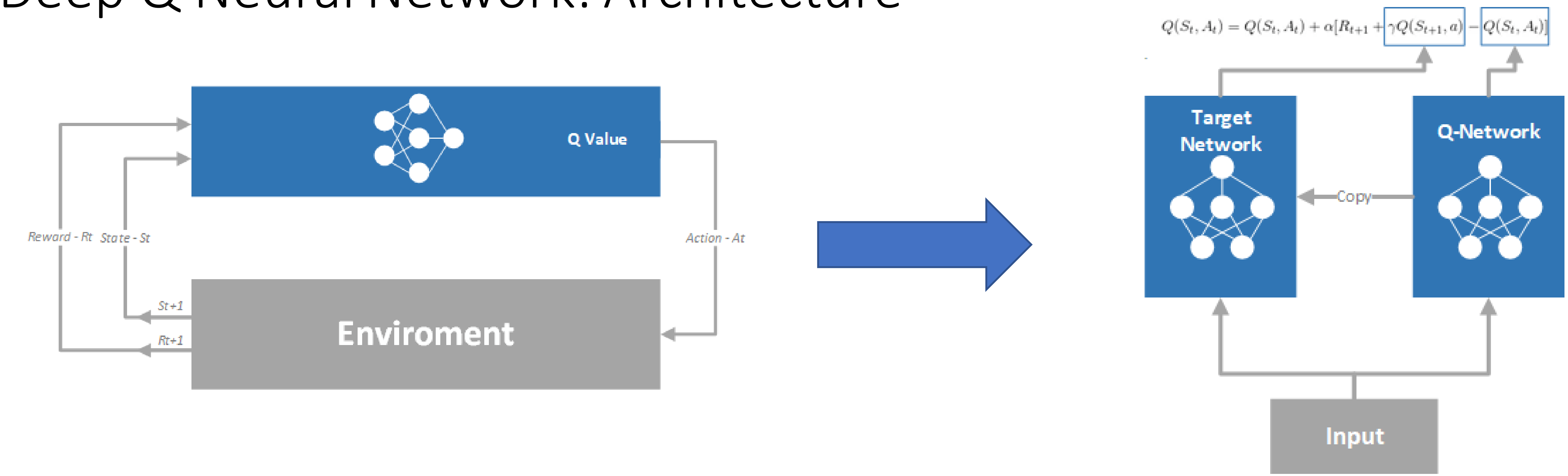








Deep Q Neural Network: Architecture



- DQN is not a Supervised Learning , We don't have labels to train
- The target is continuously changing with each iteration.
- Let's include a copy of Q Network every iteration
- The first network, which is refereed to as Q-Network is calculating Q-Value in the state St.
- The second network, refereed to as Target Network is calculating Q-Value in the state St+1.
- the Q-Network retrieves the action-values Q(St,a).
- At the same time the Target-Network uses the next state St+1 to calculate Q(St+1, a) for the Temporal Difference target.
- In order to stabilize this training of two networks, on each N-th iteration parameters of the Q-Network are copied over to the Target Network

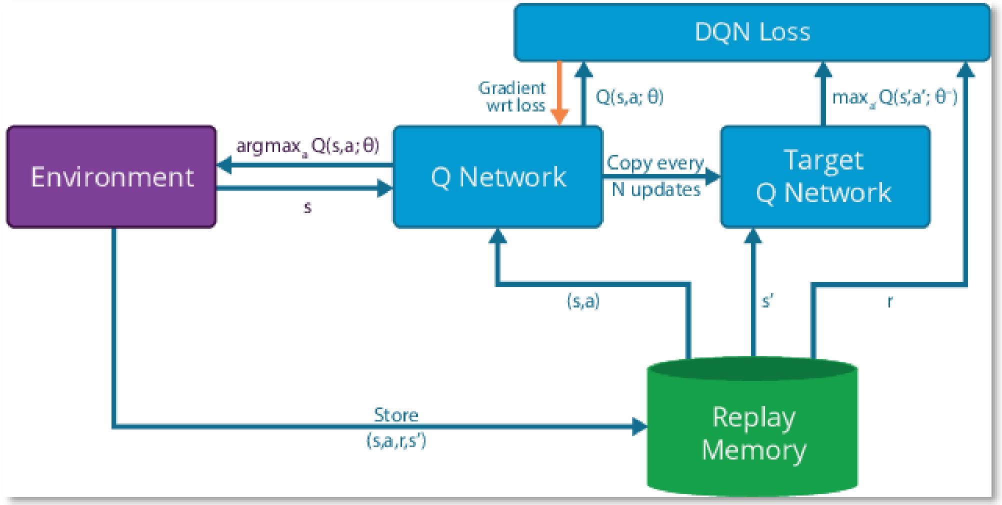
Deep Q Neural Network: Experience Replay

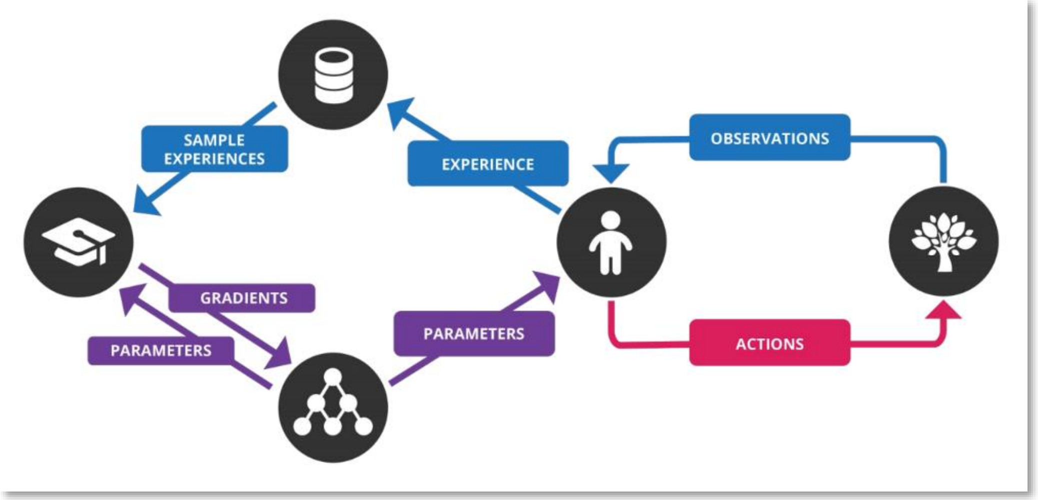
- We already mentioned that the agent, in order to train neural networks, has to **store** previous experiences.

- The naive Q-learning algorithm that learns from each of these experiences tuples in sequential order runs the risk of getting swayed by the effects of this correlation.
- *Deep Q-Learning* takes this to the next level and uses one more concept to improve performances – **experience replay**.

- This concept is used for one more reason, to stabilize training process. In a nutshell, the agent uses random batches of experiences to train the networks.

- Experience replay is the **memory** that stores those experiences in a form of a tuple ***<s, s', a, r>***:
  - ***s*** – State of the agent
  - ***a*** – Action that was taken in the state ***s*** by the agent
  - ***r*** – Immediate reward received in state ***s*** for action ***a***
  - ***s'*** – Next state of the agent after state ***s***





Deep Q Neural Network: Training Algorithm

- Bellman Equation:

$$Q(s,a) = r + \gamma max_{a'} Q(s',a')$$

l2_loss = (predicted — actual) **2

- Loss function (squared error):

$$L = \mathbb{E}[(\underbrace{r + \gamma max_{a'} Q(s',a')}_{\text{target}} - Q(s,a))^2]$$

- actual= R + γ max A` Q(S`, A`)
- R → the current immediate reward
- S` → Next state
- max A` Q(S`, A`) → max(NN output list of Q-values)
- γ → the discount factor γ → {0,1}

Initialize network Q
Initialize target network $\hat{Q}$
Initialize experience replay memory D
Initialize the *Agent* to interact with the Environment
while *not converged* **do**
  /* Sample phase
  $\epsilon \leftarrow$ setting new epsilon with ϵ-decay
  Choose an action a from state s using policy ϵ-greedy(Q)
  Agent takes action a, observe reward r, and next state s'
  Store transition $(s, a, r, s', done)$ in the experience replay memory D
  if *enough experiences in* D **then**
    /* Learn phase
    Sample a random *minibatch* of N transitions from D
    for *every transition* $(s_i, a_i, r_i, s'_i, done_i)$ *in minibatch* **do**
      if $done_i$ **then**
        $y_i = r_i$
      else
        $y_i = r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a')$
      end
    end
    Calculate the loss $\mathcal{L} = 1/N \sum_{i=0}^{N-1} (Q(s_i, a_i) - y_i)^2$
    Update Q using the SGD algorithm by minimizing the loss $\mathcal{L}$
    Every C steps, copy weights from Q to $\hat{Q}$
  end
end

Deep Q Neural Network: Applications

Retail

1. Dynamic Pricing
2. Recommendation systems

Research

1. Neural Structure Research
2. Hyper Parameter Tuning

Finance

1. Portfolio management
2. Quant Trading

Entertainment

1. Gaming platforms
2. Movie Recommendations

Manufacturing

1. Robotics and Automation
2. Process control

Automotive :

1. Self Driving cars

Questions!

Thanks for Attending my Session

Appendix

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s'

If s' is terminal:

Chasing a nonstationary target!

$$\text{target} = R(s, a, s')$$

Sample new initial state s'

else:

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \mathbb{E}_{s' \sim P(s'|s,a)} \left[(Q_\theta(s, a) - \text{target}(s'))^2 \right] \Big|_{\theta=\theta_k}$$

$$s \leftarrow s'$$

Updates are correlated within a trajectory!

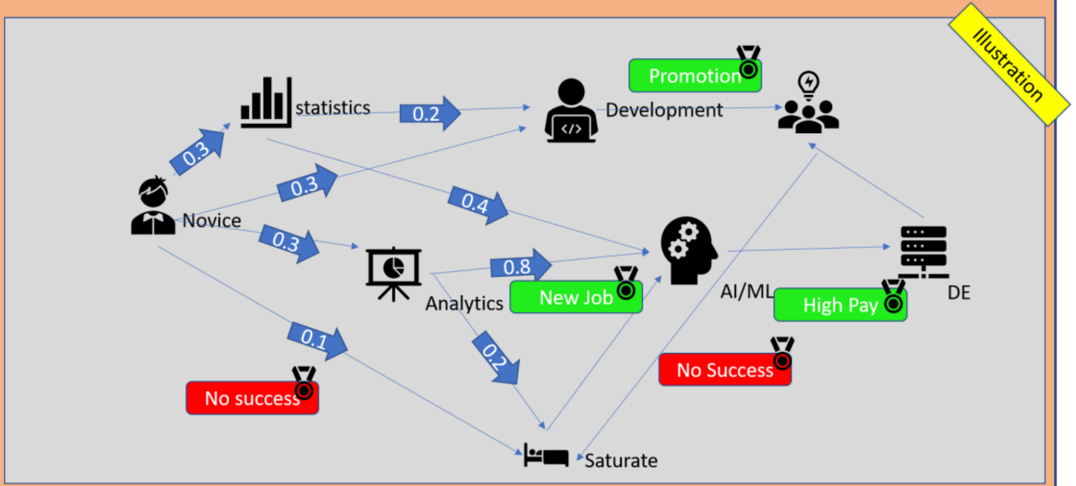
What is Reinforcement Learning? - Recap

To become successful in the industry , the person change his behavior - By learning and understanding

Here , I denote

- Person → Agent
- Each skill / position →State
- Data & AI industry → Environment
- Probability → Transition Probability
- Promotion , Failure → Reward
- Scope of Each position →value of the state
- Skill / Position transition behavior of the person →Policy

- What we need is optimal behavior / optimal policy to have more success in the environment

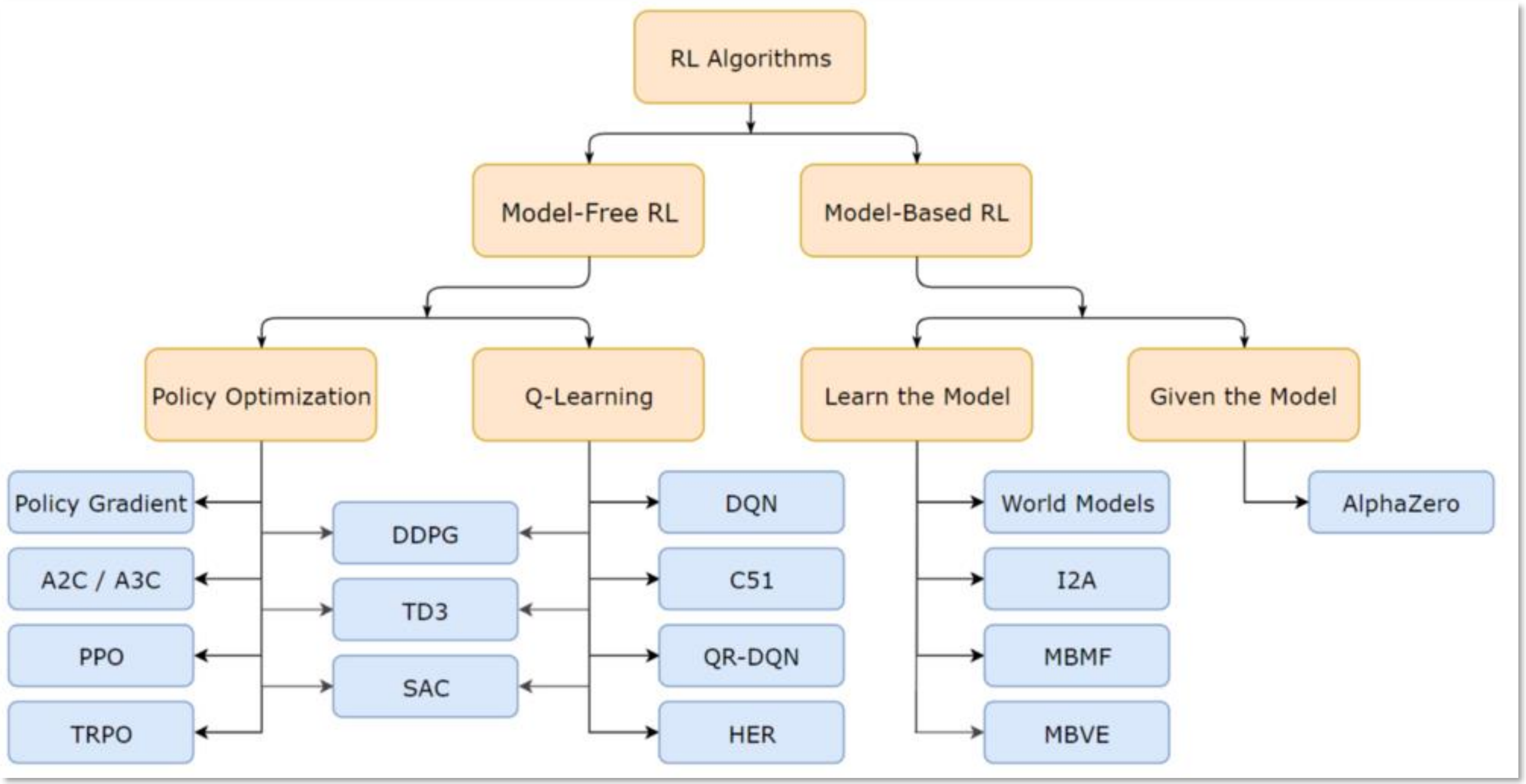


- How did I structure this process to capture these interactions ?
  - Markov Family – Helped me (Lets dive into small math part)

- By selecting correct decisions / actions in each state , the person can build his optimal policy (optimal behavior) which can give him a great success

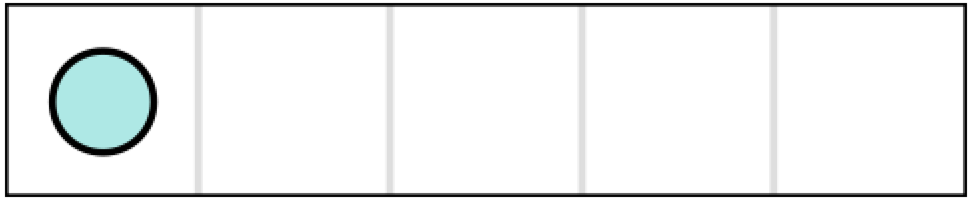
- This Selection cannot be achieved directly , its by error and trail (Learning) ---→ Reinforcement learning

How to Create that Strategy?



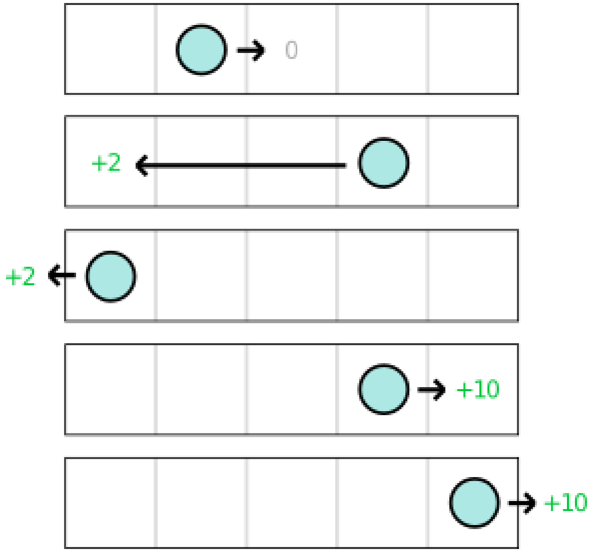
Q Learning - Q Table iteration

This is kind of a bureaucratic version of reinforcement learning. An accountant finds himself in a dark dungeon and all he can come up with is walking around filling a spreadsheet.



What the accountant knows:

- The dungeon is 5 tiles long
- The possible actions are FORWARD and BACKWARD
- FORWARD is always 1 step, except on last tile it bumps into a wall
- BACKWARD always takes you back to the start
- Sometimes there is a wind that flips your action to the opposite direction
- You will be rewarded on some tiles

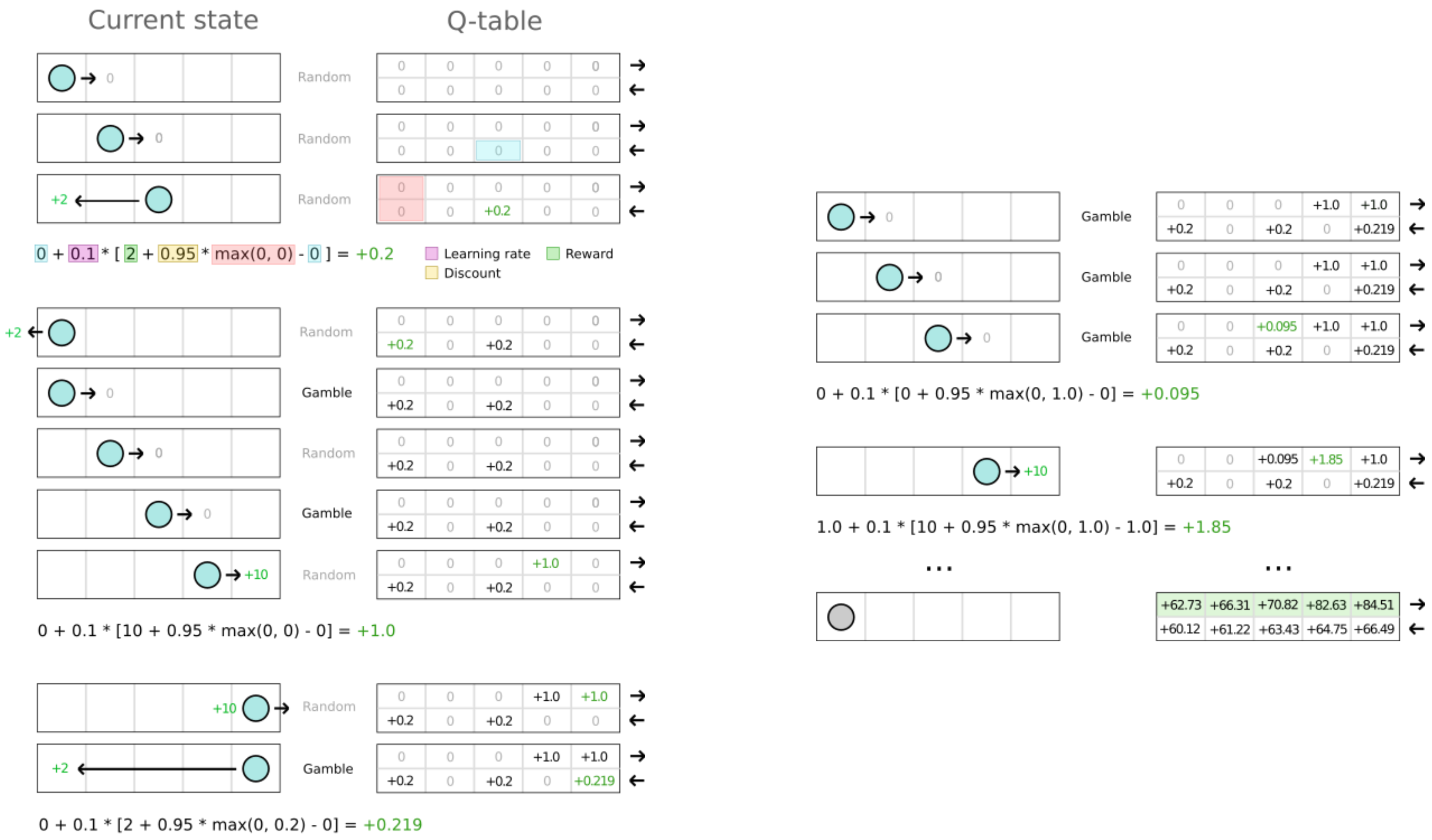


Q Learning - Q Table iteration

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \lambda \max_{a} Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

- Current Q-table value we are updating
- Learning rate
- Reward
- Discount
- Estimated reward from our next action

Q Learning - Q Table iteration



Q-Table

Q(s, a) → Q(3, 1) → +5.31

	s_0	s_1	s_2	s_3	s_4
a_0	+4.21	+3.24	+1.84	+2.33	+3.73
a_1	+2.53	+7.44	+3.34	+5.31	+6.22