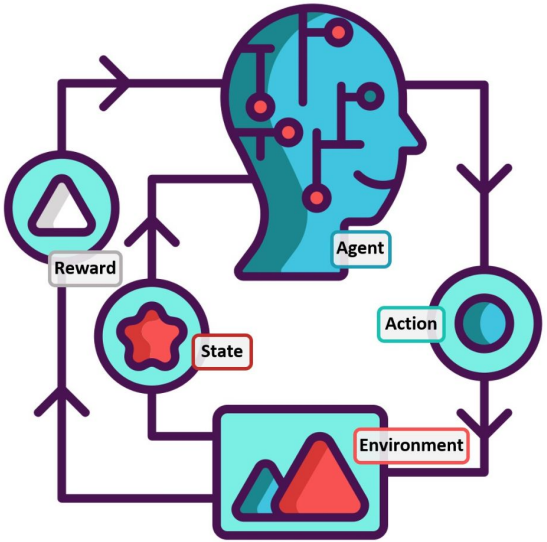
# UofT Computer Vision Club

## Reinforcement Learning




youtube.com/@uoftcv

github.com/ut-cvdg/paper-breakdown

Follow us on social media

# UofT CV is sponsored by GitHub

Sign up for the GitHub Student Developer Pack:
https://education.github.com/benefits

# Reinforcement Learning(Introduction)

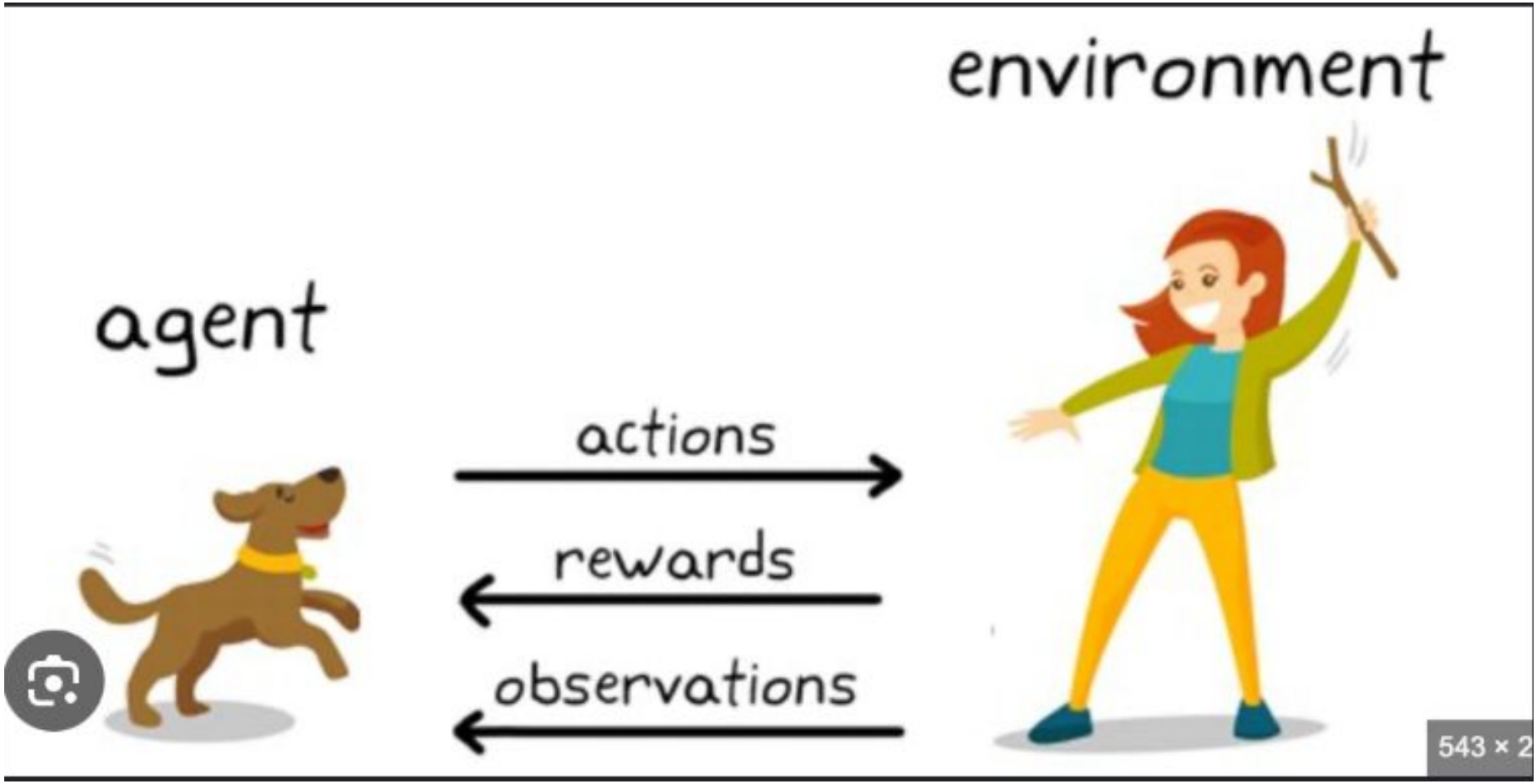
An agent

observes the world

takes an action and its states changes

with the goal of achieving long-term rewards.

Credit:CSC311 Course Slides


environment
agent
actions
rewards
observations
543 × 2

# Markov decision Process

>We work in a model called the Markov Decision Process

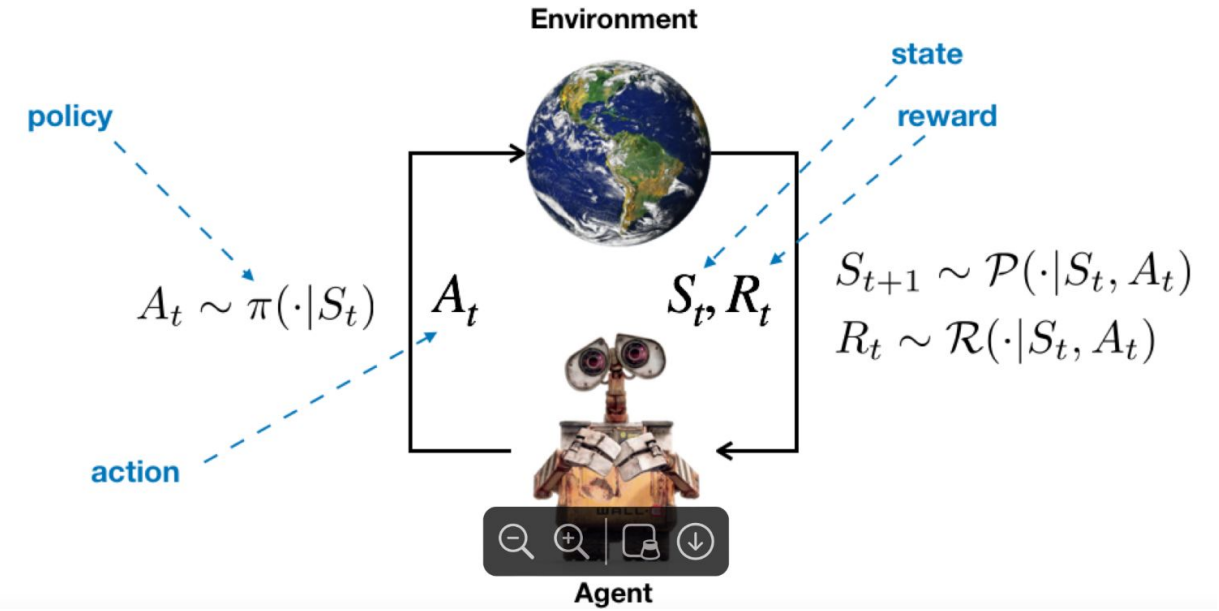
>It is a discrete time stochastic control process

> The agent is at state S.

>The decision maker then chooses an action A which puts it into the state S’. This depends on the state transition function

>It receives a reward corresponding to the action that it takes at the particular stage

# Markov Decision Process




Credit:CSC311 Course slides

# Markov Decision Process

A Markov decision process is a 5-tuple$(S,A,P_a,R_a,\gamma)$where:

> S is a set of states called the state space,

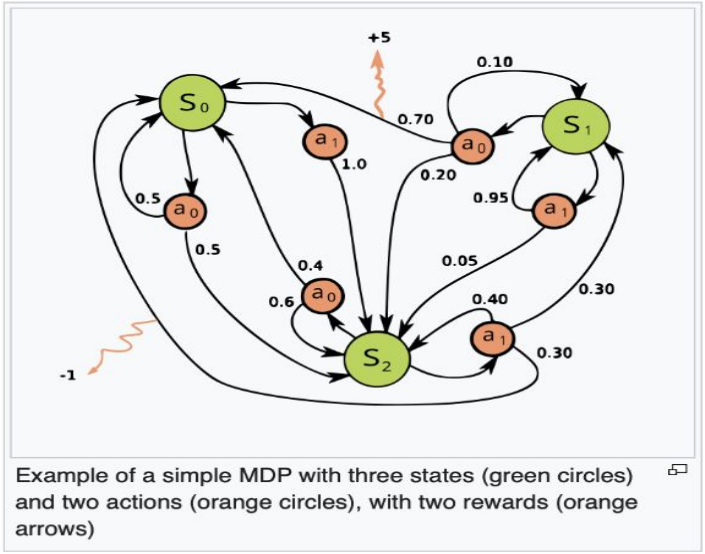
> A is a set of actions called the action space(alternatively,  is the set of actions available from state ),

>$P_a$ is the probability that action in one state at time  t will lead to next state at time t + 1 ,

>$R_a$ is the immediate reward (or expected immediate reward) received after transitioning from one state to the next state, due to an action.

>$\gamma$ :This denotes the discount factor.

# Markov Decision Process




Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows)

>This Markov Decision Process has 3 states

# Markov Decision Process(Policy)

>The action chosen by the agent at each stage is called the policy.

There are two types of policies:

1)Stochastic policies:It is a probability density function which determines the likelihood of executing each action at a particular stage.

$A_t \sim \pi(\cdot \mid S_t)$ for some function $\pi : S \to P(A)$.

2)Deterministic policies:It means that there is only one action you can take a t a particular stage.

$A_t = \pi(S_t)$ for some function $\pi : S \to A$.

# Rewards

>These are the results the the agent receives after performing  a certain action at a particular stage

>Clearly the reward is a function of the current state and the action performed at the state.

>Hence $R_t = r(A_t, S_t)$

>Two types of Rewards:

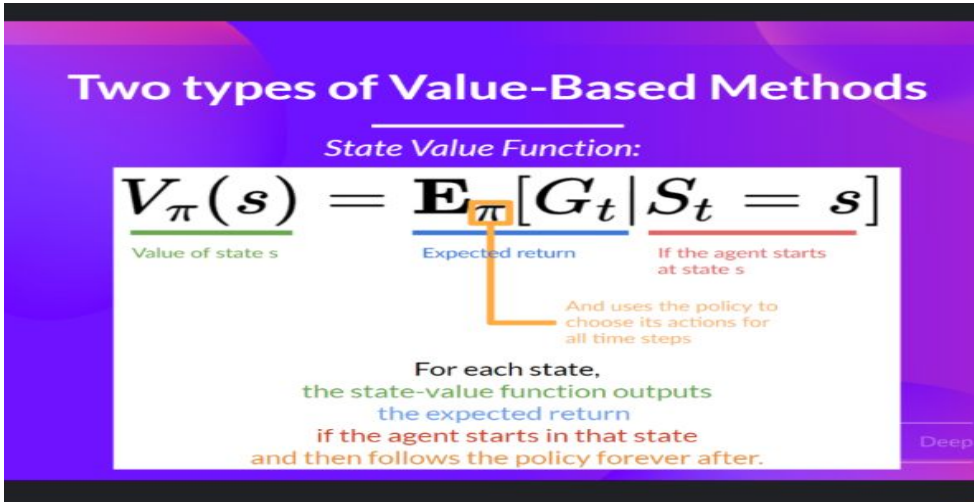
1)Discounted Rewards:G=R0+yR1+y^2R2+...

2)Undiscounted Rewards:G=R0+R1+R2+...

# Value function

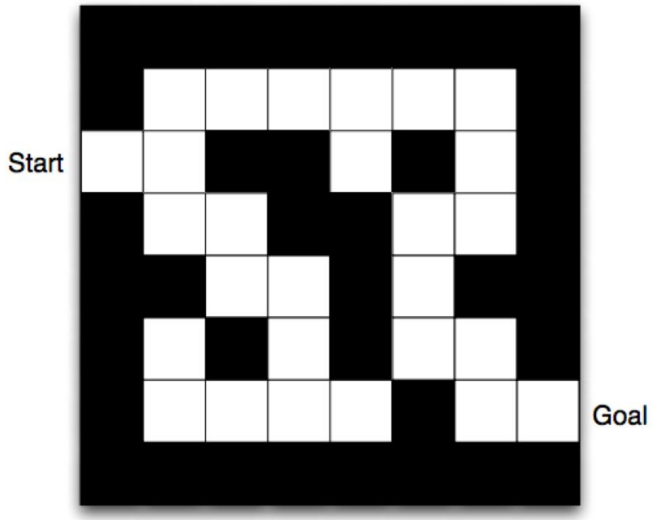
>This measures the expected return if you start in state S and follow policy

$\pi$.




$$V^{\pi}(s) \triangleq \mathbb{E}_{\pi}[G_t \mid S_t = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \mid S_t = s\right].$$

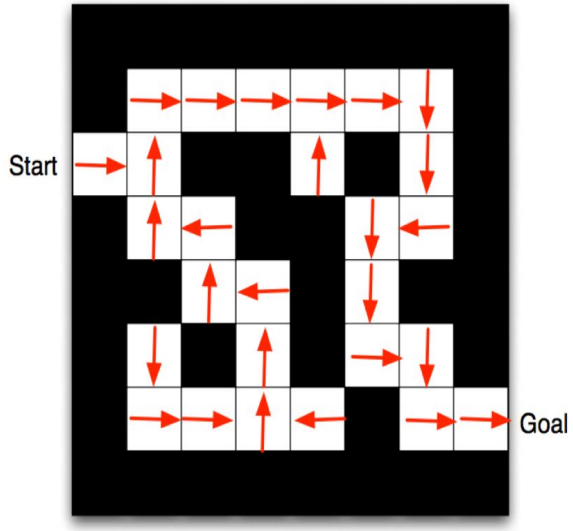
# Value Function




- Rewards: $-1$ per time-step
- Actions: N, E, S, W
- States: Agent's location

Credit:CSC311

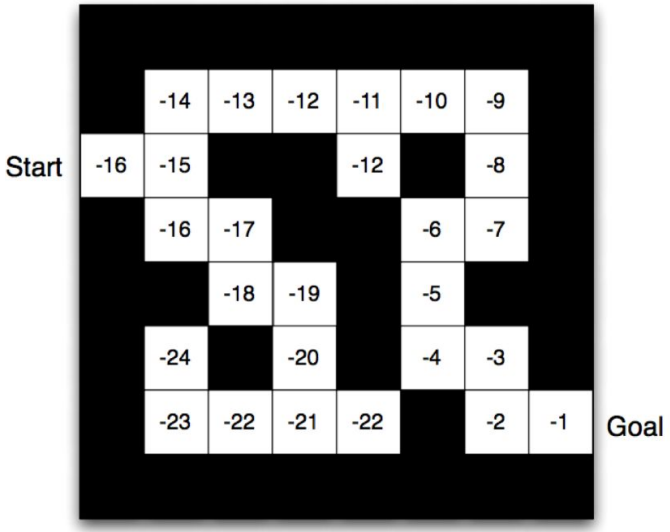
# Value Function




- Arrows represent policy $\pi(s)$ for each state $s$

Credit:CSC311

# Value Function




- Numbers represent value $V^{\pi}(s)$ of each state $s$

Credit:CSC311

# Detour into Bellman Equations

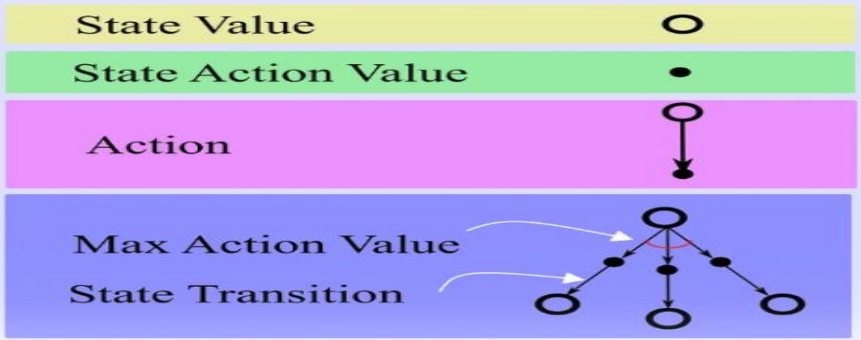
Value functions satisfy certain recursive equations known as bellman equations.

These are extremely useful in Dynamic Programming.

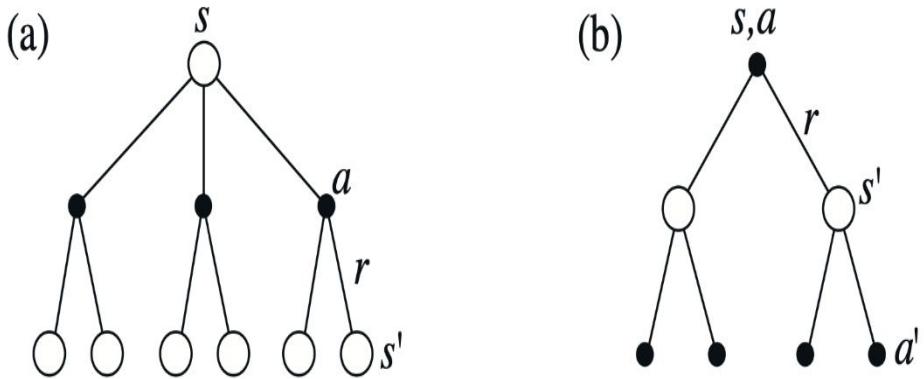
$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right], \qquad (3.12)
\end{aligned}
$$

This sums over all values

# Backup Diagram(Introduction)



# Backup Diagram



# State-Action Value Function(Q(s,a))

$$Q^{\pi}(s,a) \triangleq \mathbb{E}_{\pi}\left[\sum_{k\geq 0}\gamma^{k}R_{t+k} \mid S_t = s, A_t = a\right].$$

Q:Given the state action value function how do you think we can predict the value function for a particular state.

# State Action Value Function

- If you knew $Q^\pi$, how would you obtain $V^\pi$?

$$V^\pi(s) = \sum_a \pi(a \mid s)\, Q^\pi(s, a).$$

If you knew the value function how would you calculate the state action value function?

# State Action Value Function

- If you knew $V^\pi$, how would you obtain $Q^\pi$?
  - Apply a Bellman-like equation:

$$Q^\pi(s,a) = r(s,a) + \gamma \sum_{s'} \mathcal{P}(s' \mid a, s)\, V^\pi(s')$$

Recursive Definition

# Dynamic Programming


PLANNING

Q:Suppose you get to pick one action at a particular state what sort of action would you pick?

# Dynamic Programming

A: Pick the one with $\arg\max_a Q^\pi(s,a)$

If a policy is optimal then it must be the case that $\pi(s) = \arg\max_a Q^\pi(s,a)$

# Bellman Equation for optimal policy

$$Q^{\pi^*}(s,a) = r(s,a) + \gamma \sum_{s'} \mathcal{P}(s', \mid s,a) Q^{\pi^\star}(s', \pi^\star(s'))$$

$$= r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) \max_{a'} Q^{\pi^\star}(s', a')$$

Choosing optimal value

So if we can solve this then we can compute the optimal policy

# Policy Evaluation

Suppose that we a policy and we are trying to evaluate the policy.

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right] && (4.3) \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right], && (4.4)
\end{aligned}
$$

Now assume that we do indeed have the conditions of the environment.

Then this would be a system of equations we need to solve in order to evaluate the policy

# Policy evaluation

Instead we find an infinite sequence of values vk which converge to the required value at infinity.

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

# Updating the values

$$
\begin{aligned}
v_{k+1}(s) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big],
\end{aligned}
$$

This is how we update the values and as we tend it to infinity we will have it converge to a fixed value.

# Policy Improvement

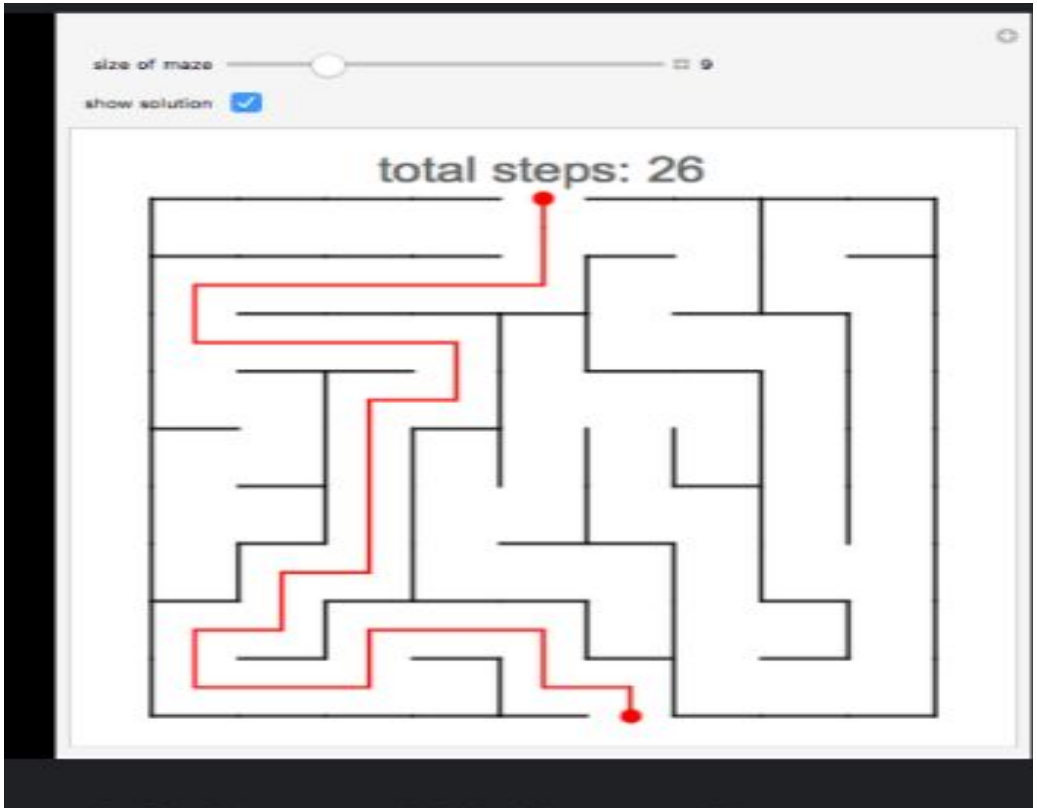
$$\begin{aligned}\pi'(s) &= \arg\max_a q_\pi(s,a) \\ &= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \qquad (4.9) \\ &= \arg\max_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big],\end{aligned}$$

We keep updating the policy as well ,till the time it does not converge to an optimal value.

# Summary

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*,$$

# Brief Introduction to Monte Carlo Methods



# Update Equation

$$V(S_t) \leftarrow V(S_t) + \alpha \left( G_t - V(S_t) \right)$$

Target

>Gt is the actual return observed from the current state

>V(St) is the current estimate of the value of the state.

Q:Why don’t we replace the value with Gt entirely?

# Temporal Difference Learning(TD)

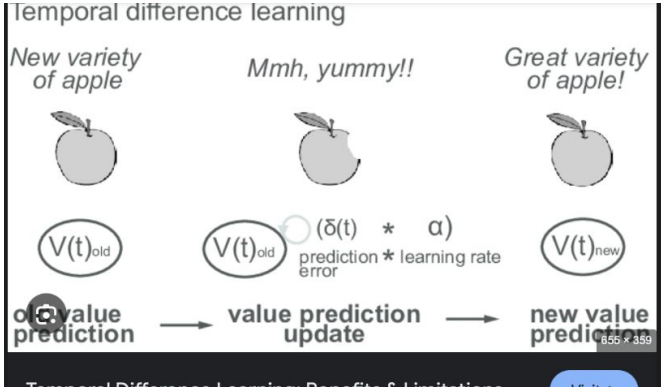
>It solves a prediction problem

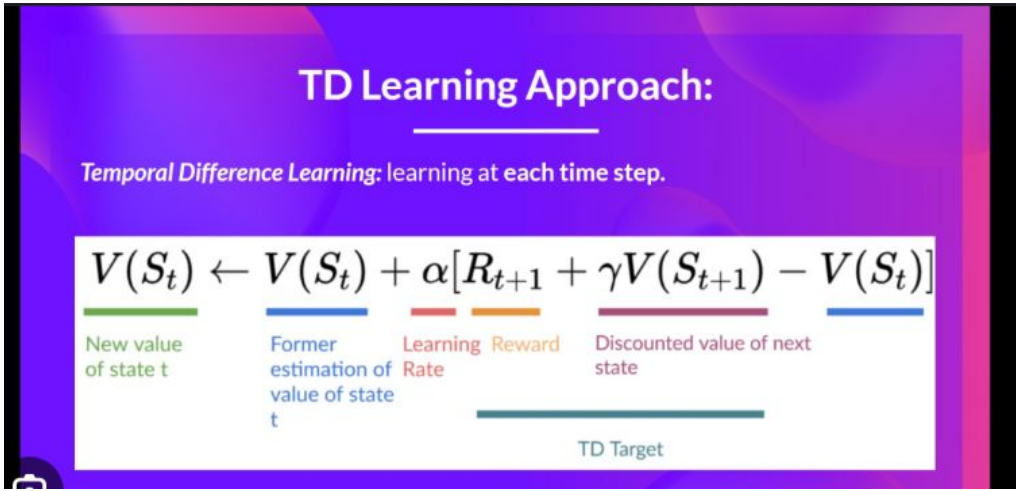
>We are trying to predict the value function for non-terminal state .

$V_{\pi}$ for a specific policy π for a specific state.

>Unlike Monte Carlo Methods TD methods need to wait only until the next step .

# TD Learning




OUR GOAL HERE IS TO USE THE VALUE AT TIME “t+1”.

# TD Learning

It combines both :

1) The sampling in Monte Carlo method

   >Estimate the value of state at “t + 1”

2) The bootstrapping of Dynamic Programming

   >The update to the current state depends on the estimated value of the next state

# In a Nutshell

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] && (6.3)\\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]\,. && (6.4)
\end{aligned}
$$

-Using the discount factor

>Monte Carlo Methods try to estimate equation 6.3

>Dynamic Programming tries to estimate equation 6.4

# Code Implementation

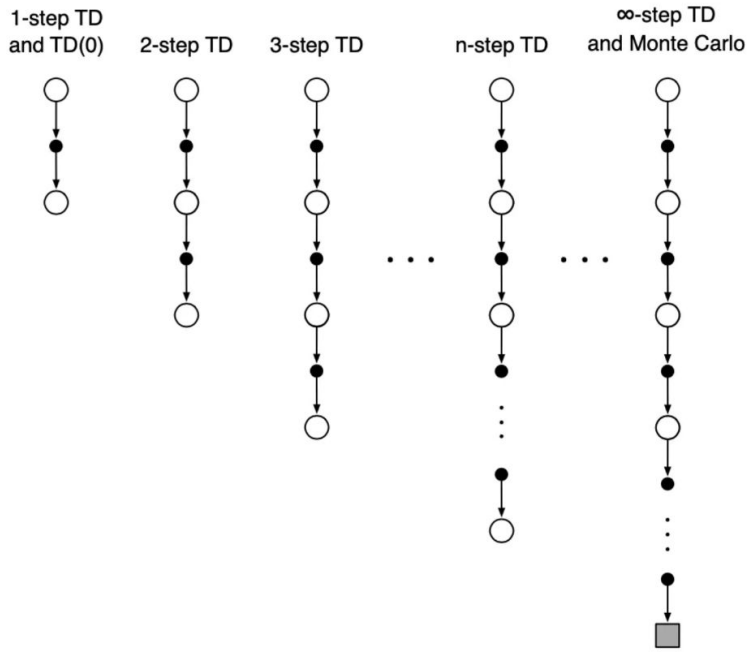
Input: the policy $\pi$ to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$; observe reward, $R$, and next state, $S'$
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until $S$ is terminal

Taking action

Updatig the Value

We stop here

# Backup Diagrams for TD




Backup diagram for n-step TD

# An example

## Driving Home

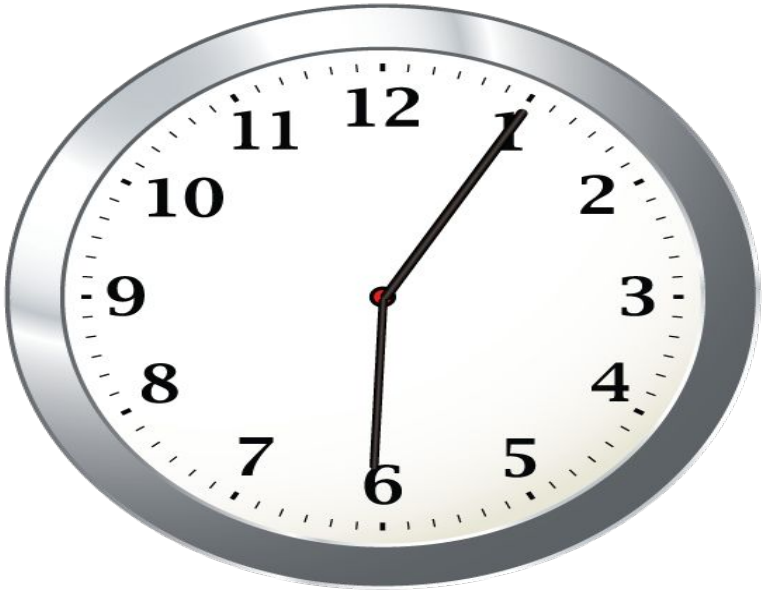
# An example

## Driving Home



dreamstime.com
ID 42786765 © Yusuf Demirci


alamy
Image ID: RXCKNR
www.alamy.com


12
1
2
3
4
5
6
7
8
9
10
11


OH DEER

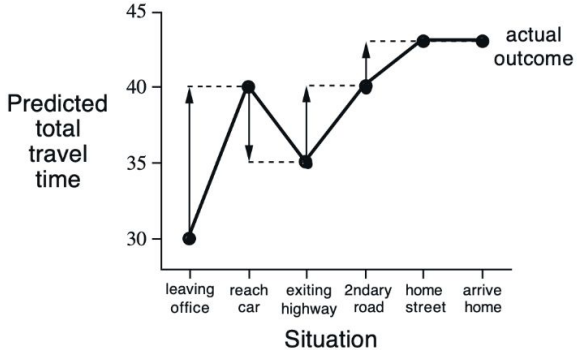
# Summary

| *State* | *Elapsed Time (minutes)* | *Predicted Time to Go* | *Predicted Total Time* |
|---|---|---|---|
| leaving office, friday at 6 | 0 | 30 | 30 |
| reach car, raining | 5 | 35 | 40 |
| exiting highway | 20 | 15 | 35 |
| 2ndary road, behind truck | 30 | 10 | 40 |
| entering home street | 40 | 3 | 43 |
| arrive home | 43 | 0 | 43 |

>Rewards in this case are the predicted time to go

# Prediction Algorithm

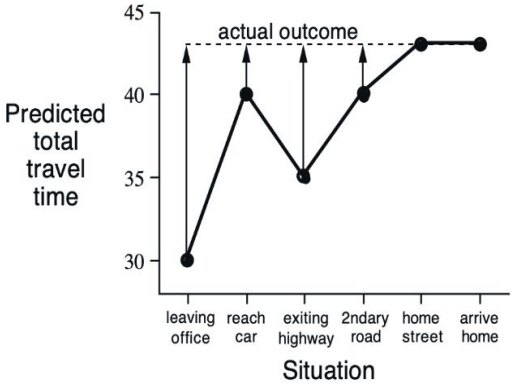
*6.1. TD PREDICTION*




>We do not wait until the final outcome for the learning to begin

>We are trying to learn based on the current predictions .

Monte Carlo Prediction




>We need to wait until the final outcome in order to begin learning because we do not know the true value of the return.

# Advantages of Temporal Difference Learning

Q:Can you name some advantages of Temporal Difference Learning in comparison to Dynamic Programming

Q:Can you name some advantages of temporal Difference Learning in comparison to Monte Carlo Methods

How about disadvantages?

# TD vs DP

>In comparison to DP methods TD does not require the entire model of the environment or the reward or the probability distribution.

>DP is much more computationally intensive in comparison to TD.

>DP methods require you to often solve a large number of equations.

>TD on other hand is not as computationally heavy, this is because TD updates its estimates incrementally as the new values are learned.

>However TD approximately guarantees the convergence to the optimal value function only under certain conditions

>DP always guarantees the convergence

# TD vs Monte Carlo

>TD methods are implemented online . They are implemented in a fully incremental fashion.

>In Monte Carlo methods the model must wait for the end of the entire episode.

>In TD we need to only wait for the next step.

>Clearly Monte Carlo methods cannot be used in cases when the length of the episode is extremely large.

>Often times Monte Carlo methods must ignore episodes in which experimental action must be taken(because they could be extremely tedious).

>Monte carlo methods on the other hand have low bias because they use the exact value of the return.

>TD methods have bias because they involve bootstrapping.

# Q-Learning

It is an off-policy Temporal Difference(TD) learning algorithm.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

$R_{t+1}$:Reward at time t + 1

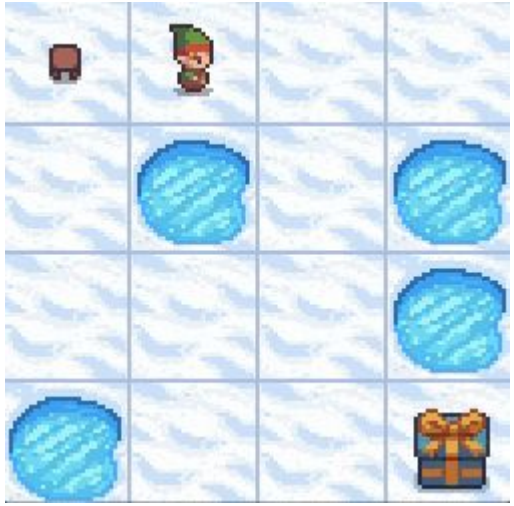
$Q(S_t,A_t)$:The Q value function

$\alpha$:Learning Rate

$\gamma$:Discount Factor

# Problem

Frozen Lake



# Objects

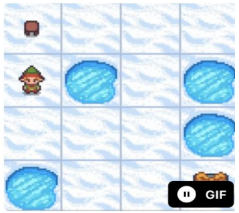
GOAL HOLES

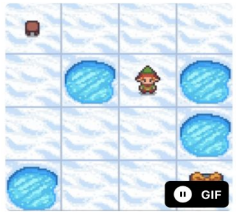
# Q-Learning(Method)

**Step 1: Define the Environment**

a)Identify the States

All possible situation the agent might encounter



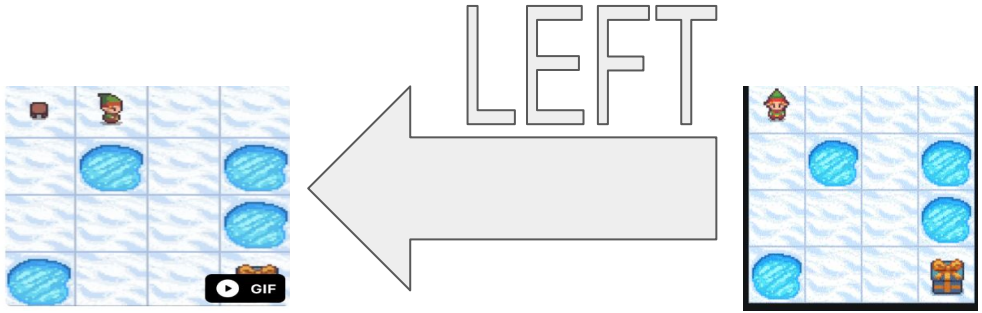




# Method

## Step 1: Define the Environment

b)Identify the Actions

All possible decisions or moves the agent can make

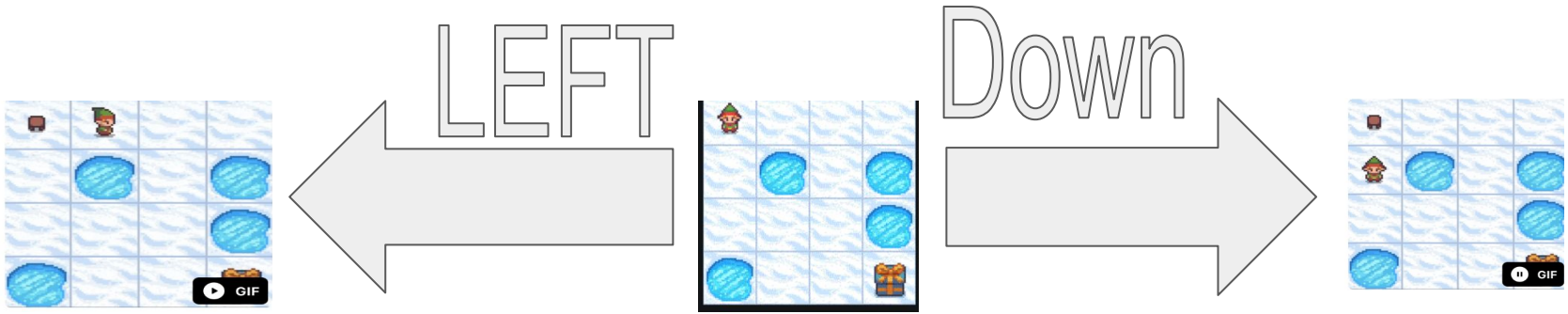


# Method

## Step 1: Define the Environment

b)Identify the Actions

All possible decisions or moves the agent can make

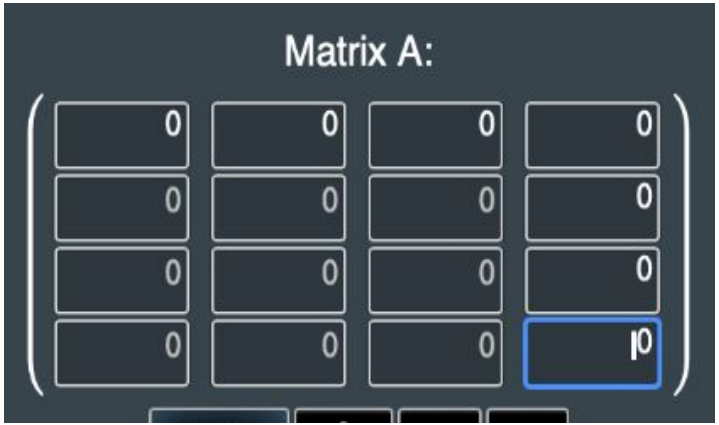


# Method

**Initialize the Q-Table**

**>**Each row corresponds to a particular state in the environment

>Each column corresponds to a particular action the agent can take.



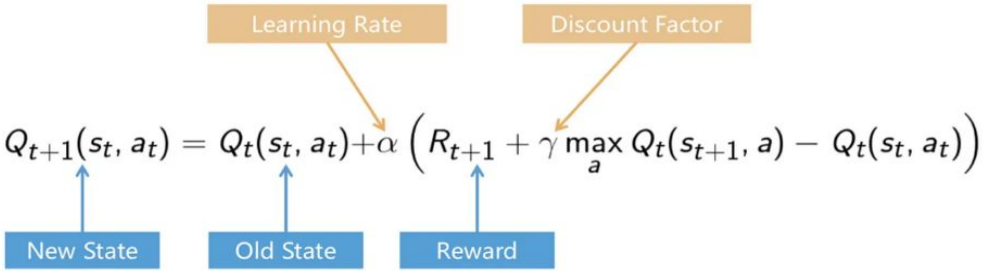
# Method

Updating the Q table

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right].$$

# Method

Updating the Q table




**Does this converge to the optimal q value?**

YES! BUT…

# WHAT ABOUT THE STATES IT MAY NOT VISIT?

# Epsilon Greedy

- Choose $A_t$ according to the $\varepsilon$-greedy policy, i.e.,

$$A_t \leftarrow \begin{cases} \text{argmax}_{a \in \mathcal{A}} Q(S_t, a) & \text{with probability } 1 - \varepsilon \\ \text{Uniformly random action in } \mathcal{A} & \text{with probability } \varepsilon \end{cases}$$

- Take action $A_t$ in the environment

# Method

Use the Q-table for decision making

>At each stage choose the Q value with the highest Q-value in the current stage.

>There are more ways to choose the next stage to visit like epsilon greedy,....

Next State to Visit if we start from (0,0)

# Code Implementation

Initialize $Q(s, a)$, $\forall s \in S, a \in A(s)$, 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., ε-greedy)

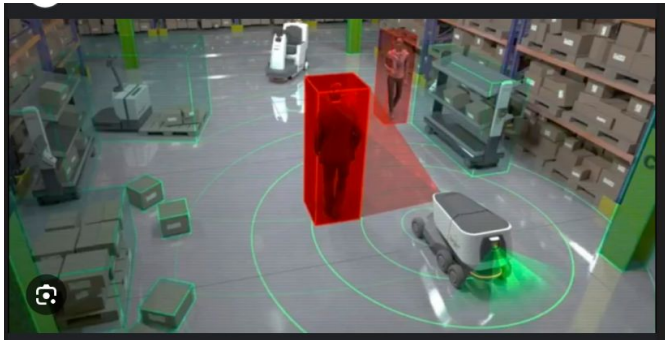
Take action A, observe R, $S'$

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

$S \leftarrow S'$;

until S is terminal

# Applications of Q-learning

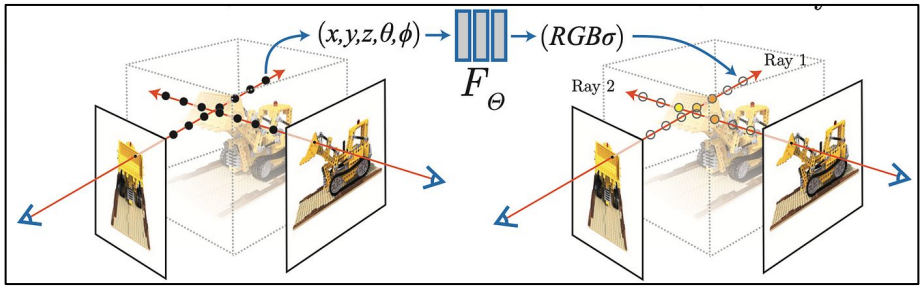


Slides on github.com/ut-cvdg/paper-breakdown

# Thank You

Later: NeRF






Follow us on social media