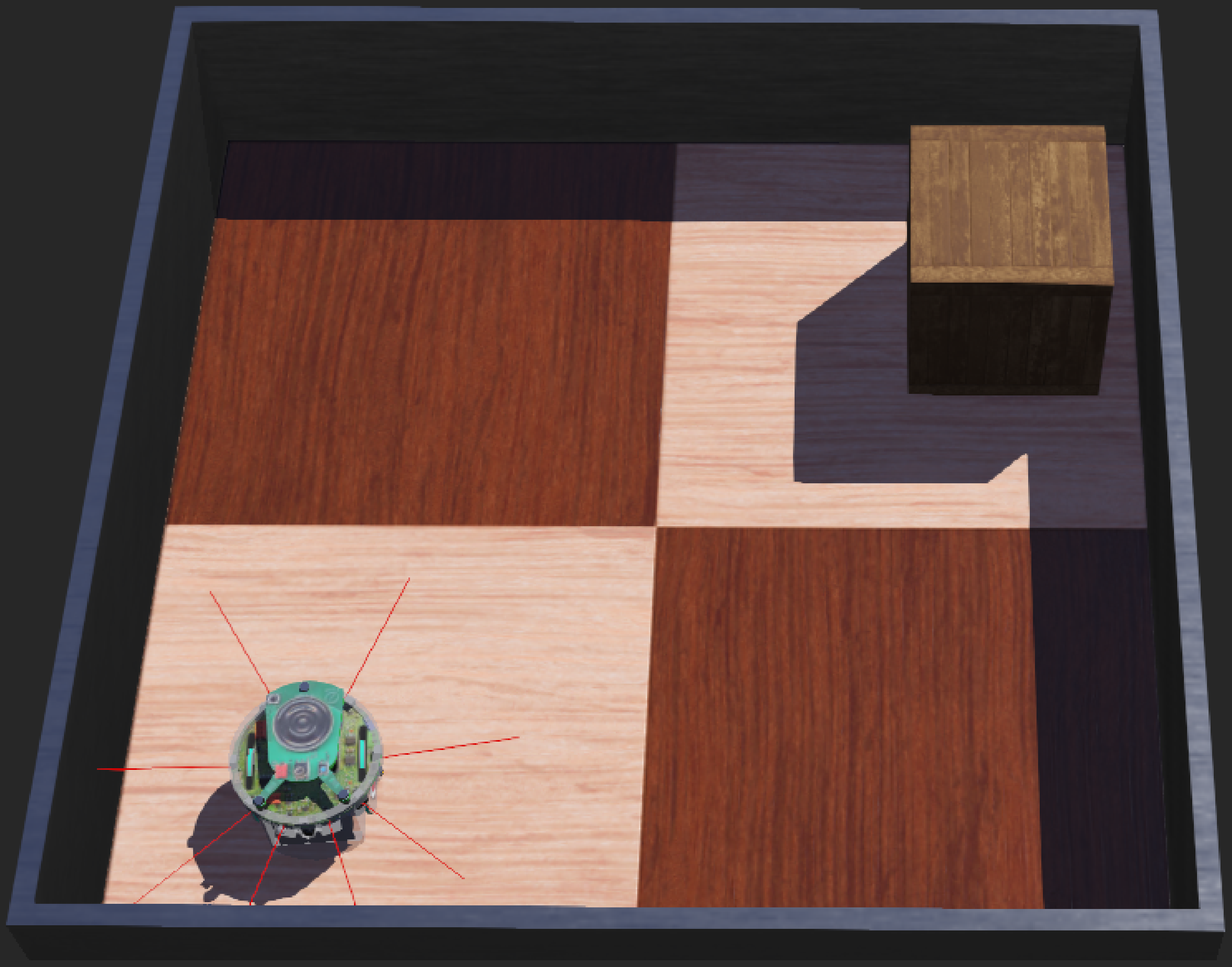
REACH TARGET LOCATION (USING DEEP DETERMINISTIC POLICY GRADIENT)



Supervisor: Francisco Azeredo
By: Siddarth Balasubramani, Akshay Ratnawat

Problem Statement

Objective is to teach robot to find and reach the target object in the minimum number of steps and using the shortest path and avoiding any obstacles such as humans, walls etc usinf reinforcement learning algorithms.

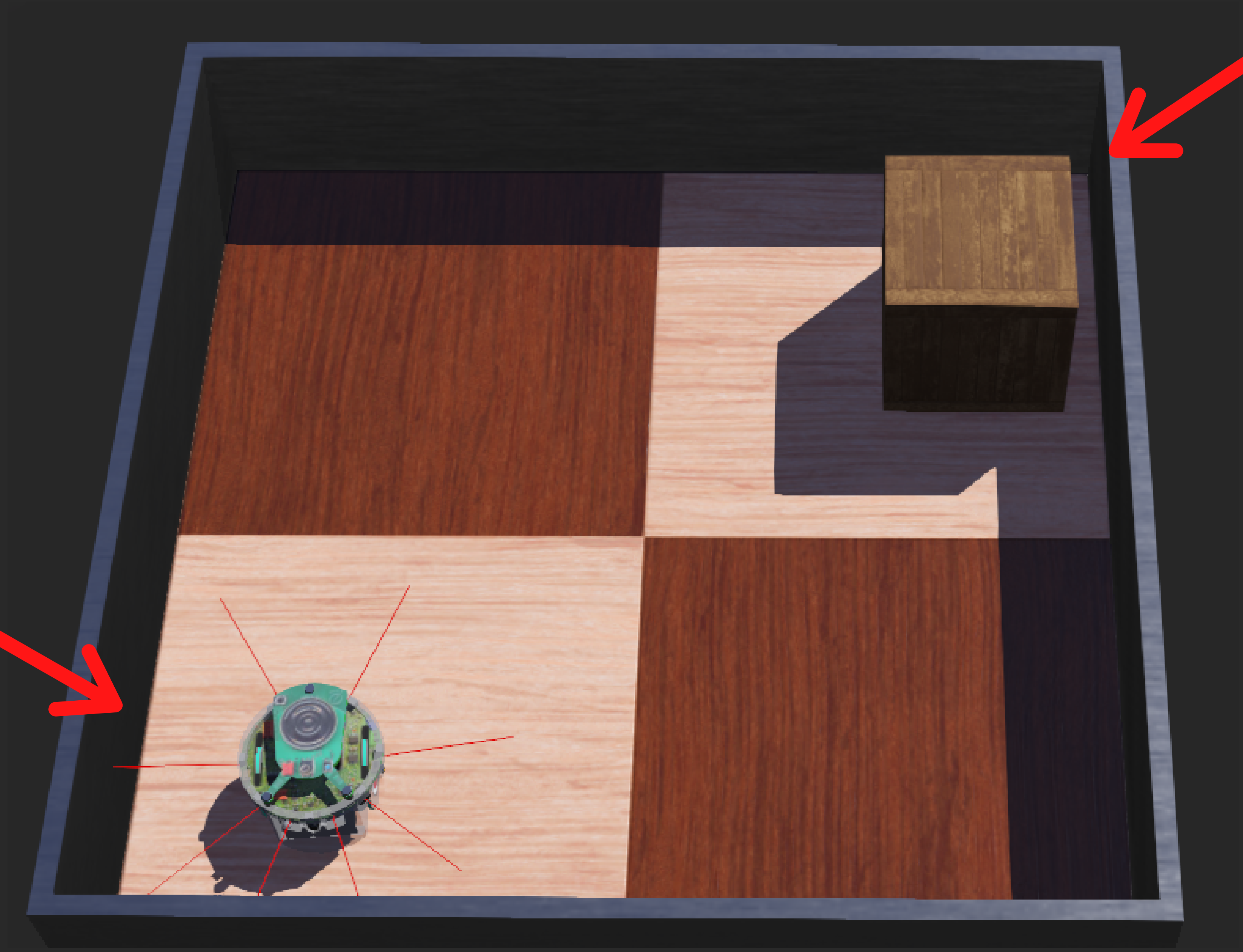
How Reinforcement Learning can help

- Most robots used in factories today are powerful but blind.
- They are programmed to do specific tasks repeatedly with high precision and speed, but they are not able to adapt to changes in the environment or handling variables. For eg: Robot arms often lack the ability to grasp an object if it is moved a few inches.
- We can use traditional computer vision to solve this problem but it requires pre-scanning and building computer-aided design (CAD) models for all items beforehand with expensive 3D cameras. The process is time-consuming and inflexible.
- Even if there was a box with exactly the same shape, but a different size, a robot couldn't automatically recognize the differences and figure out how to grasp the new item.
- Order picking represents more than 40% of operational costs across most warehouses. While tasks are repetitive, they involve some variation with products and packaging design, making it impossible to hard-program robots in advance.
- Deep reinforcement learning (DRL) can unleash the full potential of robotics, giving them the ability to recognize and react to their surroundings and handle product variations.

Environment

Target Object/Location

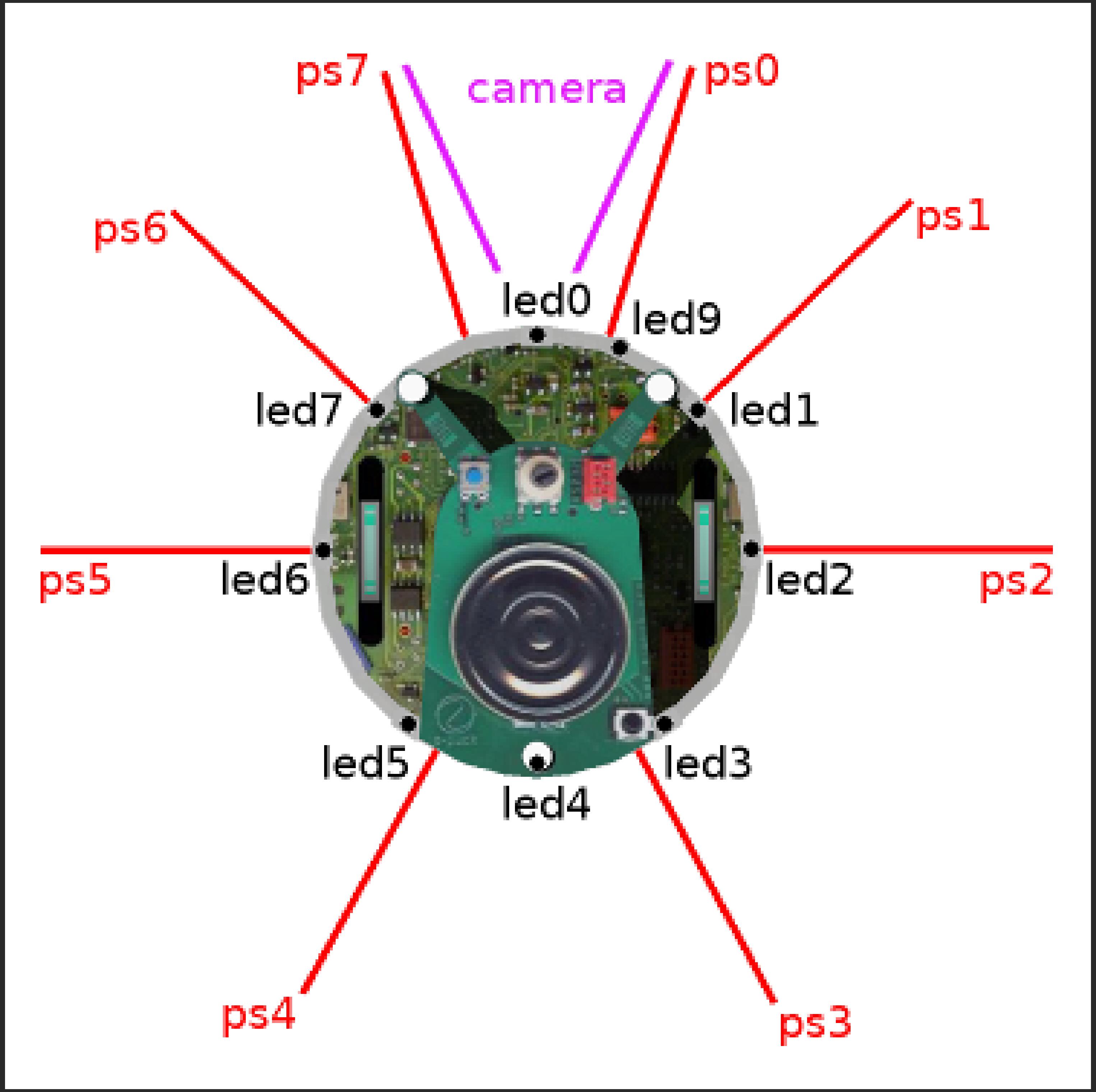
E-Puck Robot



- The Coordinates of the Target Location and the current positon of the Robot is known.
- The environment is partially observable to the robot as the sensors of robot gives the local information of the agent in the environment and not complete information .
- Since the environment is partially observable we do not know the exact current state, thus we track the state alias.
- Start state of the robot is same in all the episodes.

(**Note:** Coordinates are know with respect to environment. This project does not use GPS tracking system, or any pre-defined directions to the robot through tracks etc.)

E-Puck Robot



- It is a differential wheel robot i.e. both the left and right wheel motors can be controlled differentially.
- 9 LED Lights
- 8 Distance Sensors
- Front Camera
- Infra Red Emitters and Receivers
- Accelerometer

Environment Variables: States and Actions

Observation/State Space

- List of 10 values at each time step.
- [Distance_Sensor 1 - 8, Distance from Target, Angle from Target]
- We are taking a normal Euclidean Distance.
- Angle of Robot to Target is calculated using Trignometric Calculation given the Robot Angle and Coordinates of the Robot and Target.

Action Space

- The agent has two continous actions.
- Both the actions are just how much speed we should move.

Action 1: Gas (for moving forward)

- It defines the speed at which it wants to move. Value between -1 and 1

Action 2: Turning (Gas for turning)

- It defines the speed for turning. Its between -1 and 1.

Final Action:

Left Motor Speed: Gas + Turning

Right Motor Speed: Gas + Turning

Environment Variables: Reward

Condition	Rewards
Steps > 6000	-10
If Distance from Target is < 0.1 units	10
If abs(action) > 1.5 and Steps > 10 (Robot will keep on rotating for 10 steps)	-1
If Distance from target keeps on increasing for more than 10 steps	-1
If any of the Distance sensor values is more than 500 for more than 10 steps	-1
If any of the Distance sensor values is more than 100 for more than 10 steps	-0.5

Final Reward = Reward - (Total_Steps/6000)

(This is to penalise the robot for taking more time/steps to reach the target location)

Environment Variables: Terminal/Reset States

We Reset the Environment under following conditions:

- If the steps > 6000.
- If the Robot hits any of the walls or obstacles (The target object is also an obstacle).
- If the distance between the robot and Target is equal to 0.1 units

Supervisor and Robot Controller

We use two controllers while training the same robot.

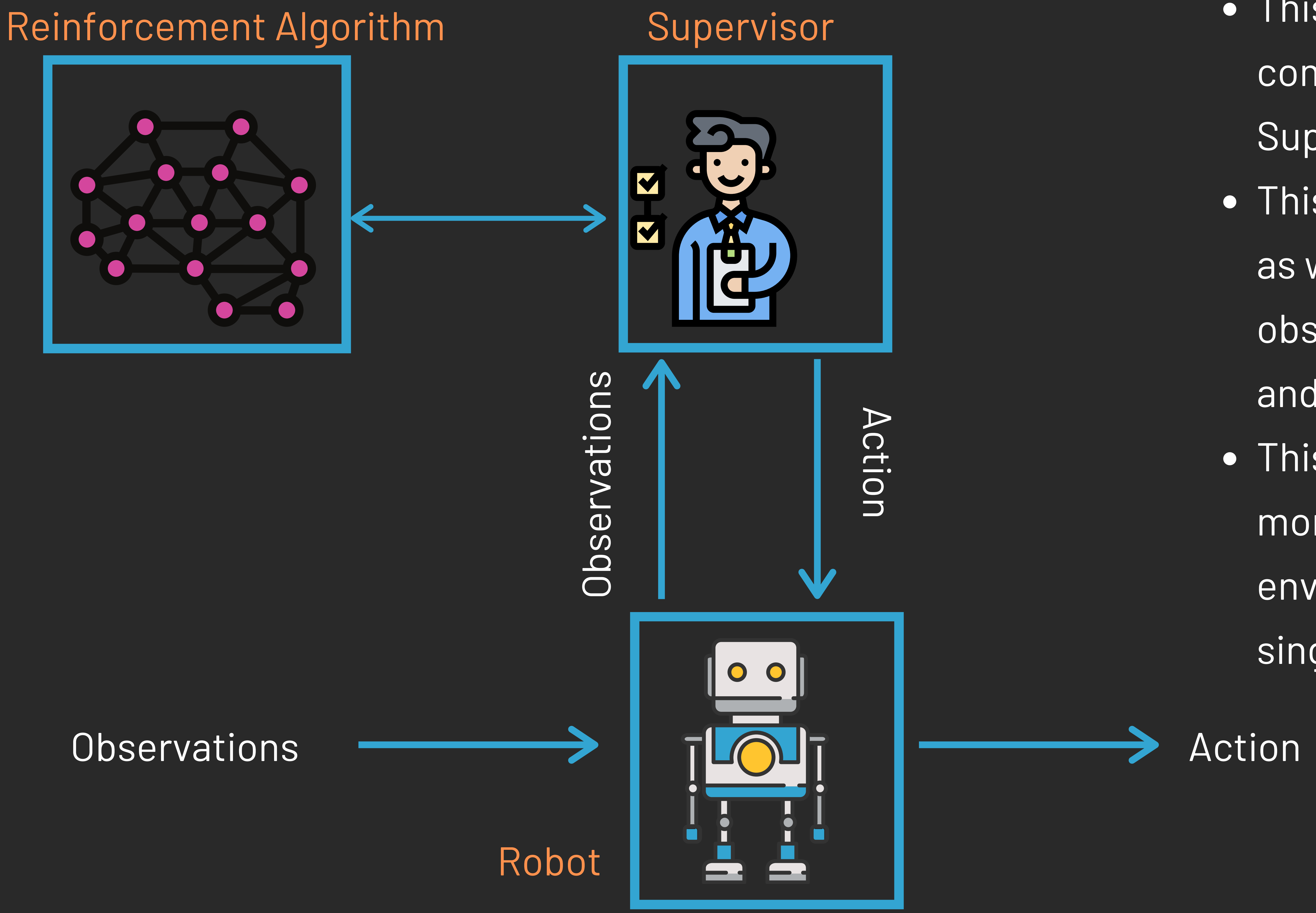
Robot Controller

- It has access to fuctions related only to the robot.
- It helps in data collection only of the robot and not its state in the environment with respect to other objects.

Supervisor Controller:

- It gives more control over the whole environment and replaces the human actions, such as measuring the distance between the two objects, finding the location of the object etc.
- It also helps in controlling the simulation of the environment, like when to terminate the simulations etc.
- This distinction is necessary to ensure that many of these functions may not be transferred to the real robot.

Emitter - Receiver Framework



- This framework is required for communication between the Supervisor and the Robot.
- This also allows in faster training as we can connect multiple robots observations to a single supervisor and train them together.
- This is also helpful when we have more than one robot in same environment, collaborating to do a single task

Deep Deterministic Policy Gradient (DDPG)

DDPG is a **Model Free, Off Policy, Policy Gradient** Algorithm

- Deep Deterministic Policy Gradient (DDPG) is an algorithm which concurrently learns a Q-function and a policy.
- It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy.
- It learns both Q-Function and policy using Deep Neural Networks.

Deterministic Policy Gradient (DPG):

- Work over continuous space
- Policy Gradient (Actor Critic)

+

Deep Q-Learning (DQN):

- Learning-Based
- Target Network
- Replay Buffer
- Does not work over continuous space

Note: *On-policy methods attempt to evaluate or improve the policy that is used to make decisions. In contrast, off-policy methods evaluate or improve a policy different from that used to generate the data

Deterministic Policy Gradient (DPG)

- Finds Deterministic Policy
- Applicable to continous action space

In Q- Learning we find the deterministic policy by :

$$\pi^*(s) = \boxed{\arg\max_a} Q^*(s,a) \quad \forall s \in \mathbb{S}$$

Problem : In large discrete action space or continuous action space, we can't plug in every possible action to find the optimal action!

Solution : Thus we learn a function approximator for argmax, via gradient descent

- We define a set of parameters θ to parametrize this policy π_θ.
- Where J(θ) is the objective function which is expected reward following a parametrized policy, and we would want to maximize it.

$$\begin{aligned}
&\textit{Policy} : \pi_\theta \\
&\textit{Objective function} : J(\theta) \\
&\textit{Gradient} : \nabla_\theta J(\theta) \\
&\textit{Update} : \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)
\end{aligned}$$

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

Alpha is the Learning Rate or Step Size

Policy Gradient Theorem

So we have our Objective Function as : (Maximizing the total expected rewards over a policy)

- We want to find the gradient of this function with respect to θ.

$$J(\theta) = \mathbb{E}[\sum_{t=0}^{T-1} r_{t+1}]$$

The Policy Gradient Theorem: The derivative of the expected reward is the expectation of the product of the reward and gradient of the log of the policy π_θ.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) r\right]$$

We can easily replace the instaneous reward with the long term Q (s,a) values

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \boxed{\pi_\theta(s,a)\ Q^{\pi_\theta}(s,a)}\right]$$

DDPG Learns both these using Deep Learning.

We can get rid of the expectations by sampling large number of trajectories (Monte Calro simulations) and average them out.

How Policy is determined?

- To determine the policy to be tested we use a Gaussian Policy.
- We choose an action from a Normal Distribution with a certain mean and variance.

Problems with Monte Carlo Policy Update

We saw that we can replace the Expectation in the below equation using large number of Monte Carlo simulations.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ Q^{\pi_\theta}(s, a)\right]$$

Problems:

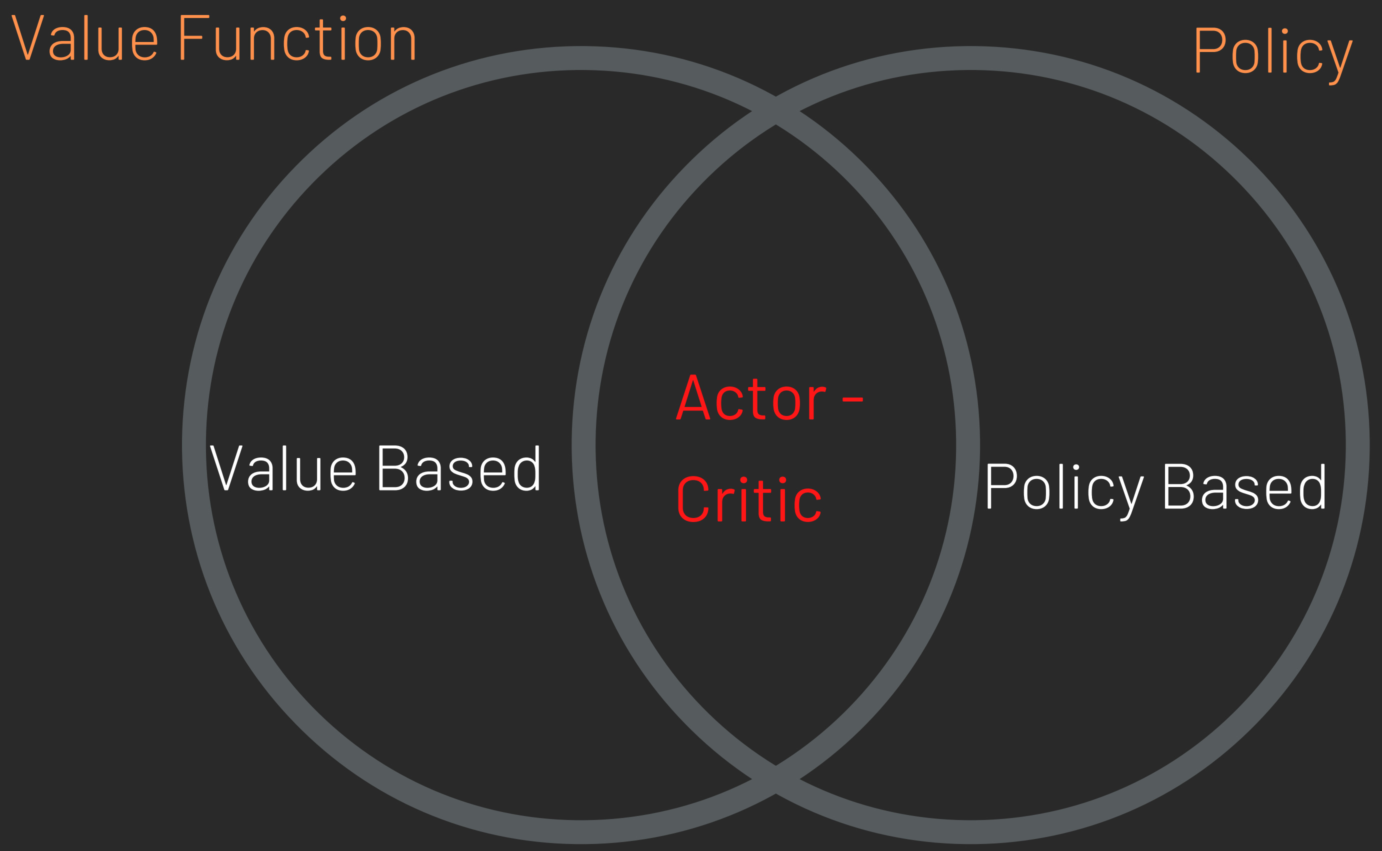
- Using MC updates can lead to high variability in the log of the policy distribution and cumulative reward values, because each trajectories during training can deviate from each other at great degrees.
- High variability in log probabilities and cumulative reward values will make noisy gradients, and cause unstable learning and/or the policy distribution skewing to a non-optimal direction.
- In MC trajectories may have a cumulative reward of 0. The essence of policy gradient is increasing the probabilities for "good" actions and decreasing those of "bad" actions in the policy distribution. But when we have a cumulative reward of 0 we cannot distinguish between both "goods" and "bad" actions.

Thus, Monte Carlo simulations make it hard to evaluate our actions.

Actor Critic Method: To evaluate our Actions

We reduce the variance we see in Monte Carlo Updates using the Critic Network

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^{\pi_\theta}(s,a)\right]$$



We have two networks:

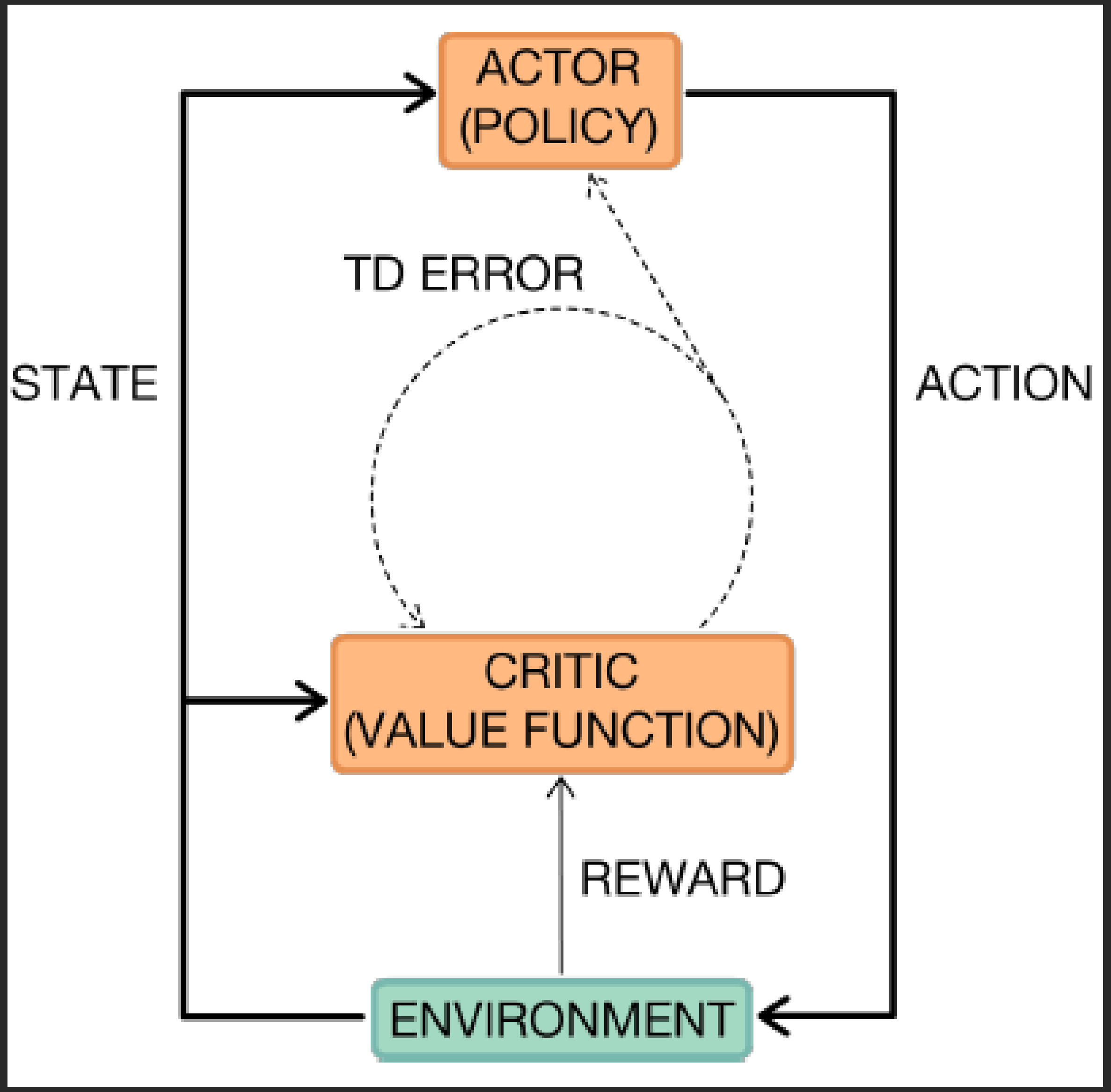
Actor Network:

- Our Network outputs deterministic action values based on current state features/observations.
- We add a gaussian noise to the actions (1% of the times) for the explore - exploit of the environment.

Critic Network:

- Evaluates the action state pairs.
- Critic here is solving the problem of policy evaluation i.e. how good a policy is for the current parameters of the network.

Actor Critic Architecture



- The “Critic” estimates the value function.
- The “Actor” updates the policy distribution in the direction suggested by the Critic (such as with policy gradients).

In a simple TD(0) algorithm of Actor Critic, at every step in our episode:

- We sample our transitions (S, A, S', R)
- Actor takes action A according to our policy (Gaussian Distributed).
- We get a TD error (Value after the step - Value before the step)
- Update our Critic in the direction of the TD Error
- Then we update the actor policy based on feedback by critic as to which actions are good and which are bad.

DPG + DQN = DDPG

DDPG, learn both the policy and the value function in DPG with neural networks, with DQN tricks!

Deterministic Policy Gradient (DPG):

- Formulates an update rule for deterministic policies, so that we can learn deterministic policy on continuous action domain
- Model-Free,
- Actor-Critic

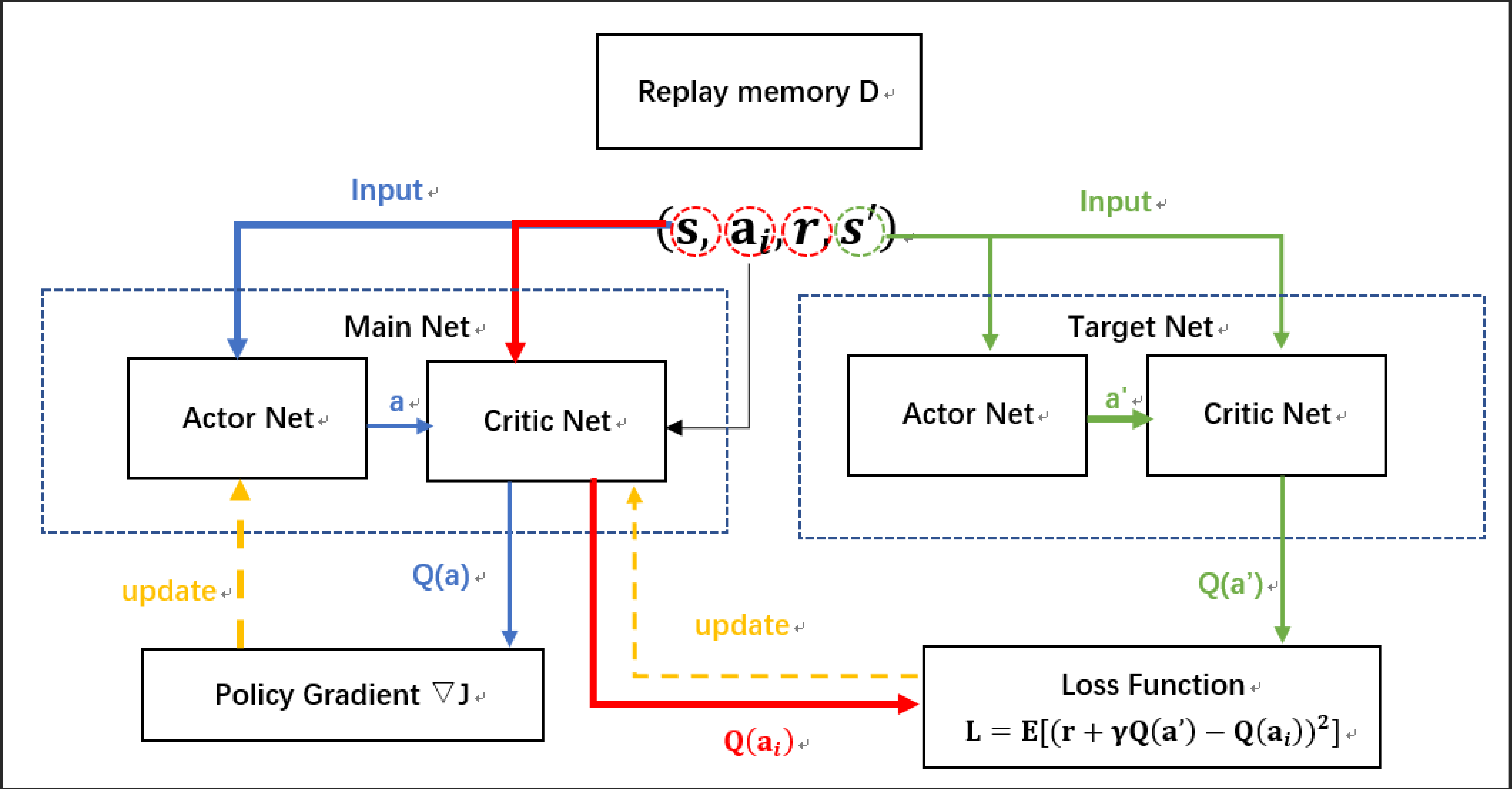
Deep Q-Learning (DQN):

DQN: Enables learning value functions with neural nets , with two tricks:

- Target Network
- Replay Buffer - Off-Policy

DDPG is applying Target Network and Replay Buffer tricks on Actor-Critic Algorithm

DDPG Architecture



Before Training

A random walk was set up as the default controller for the robot.

During Training (First 15 Episodes)

- Hitting any obstacle is a terminal state. Thus whenever robot hits the wall it termiantes and starts back from the inital position.
- Robot is having a continous action space which makes it difficult for the robot to learn as it can try multiple options in the same direction.
- But still robot learns somewhat and try to avoid the right wall.

During Training (100-1000 Episodes)

- After 100 Episodes it learns what actions are necessary for it to not hit the wall.
- Then it learns which turning actions are appropriate atleast for the initial states and comes out of that state of hitting the wall.
- It now also learns how to avoid the right wall to a certain extent.

Testing (After 5000 Episodes of Training)

There are two continous actions available to robot:

1. **Turning:** Optimum turning action is:
    a. To turn almost 180 degrees,
    b. Reach the diagonal position
    c. Then just move straightforward along the diagonal.
2. **Speed:** Optimum speed should be the max speed with the above turning actions.

Learning Takeaways:

- Robot learns the optimum turning action as it turn 180 degrees, reach the diagonal position.
- But then instead of taking the diagonal path it learns some curvy path along the diagonal.

As per various sources to solve an almost similar problem it can take upto 4 Million* Episodes of training.

Ref: *https://www.davidsilver.uk/wp-content/uploads/2020/03/pg.pdf

Thank You

Open for any questions.

References:

1. Everything You Need to Know About Deep Deterministic Policy Gradients (DDPG): https://youtu.be/4jh32CvwKYw
2. https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver
3. https://declanoller.github.io/2019/03/27/training-a-real-robot-to-play-puckworld-with-reinforcement-learning/
4. https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f
5. https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b
6. https://julien-vitay.net/deeprl/PolicyGradient.html#sec:policy-gradient-methods
7. https://towardsdatascience.com/policy-gradients-in-a-nutshell-8b72f9743c5d
8. https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.cellstrat.com%2F2020%2F03%2F19%2Frl-with-actor-critic-methods%2F&psig=AOvVaw04h3Sn2FHLmKdjRo79Ssi1&ust=1622060646024000&source=images&cd=vfe&ved=0CA0QjhxqFwoTCljfgPfU5fACFQAAAAAdAAAAABBu
9. https://www.programmersought.com/article/17164275737/
10. https://www.pair.toronto.edu/csc2621-w20/assets/slides/lec3_ddpg.pdf
11. https://hackernoon.com/why-is-warehouse-automation-so-important-for-ai-watchers-6f802e70271d