EasyChair Preprint

№ 9532

Metaverse in InterPlanet Internet: Reinforcement Learning Implementation of Time-Dependent Machine Learning Model to Make Robots Fit for Space Applications

Poondru Prithvinath Reddy

January 3, 2023

*

Metaverse in InterPlanet Internet: Reinforcement Learning Implementation of Time-dependent Machine Learning Model to Make Robots fit for Space Applications

Poondru Prithvinath Reddy

ABSTRACT

The interplanet internet is a conceived computer network in space, consisting of a set of network nodes that can communicate with each other. These nodes are the planet's orbiters (satellites) and landers (e.g. robots, autonomous machines, etc.) and the earth ground stations, and the data can be routed through Earth's internal internet. As resource depletion on Earth becomes real, the idea of extracting valuable elements from asteroids or using space-based resources to build space habitats becomes more attractive, one of the key technologies for harvesting resources is robotic space mining(minerals, metals, etc.,) or robotic building of space settlement. The metaverse is essentially a simulated digital environment mimicking the real world. The metaverse would be something very similar to real world planetary activities where users(space colonies or internet users on Earth) interact with overlaying objects represented by robots, drones, etc. for real-world planetary activities like space mining, building space settlements, etc. in a completely virtual manner. In this paper, we use information about different time steps as represented by the observation matrix for the presence of the robots in a planetary environment by encoding the robots presence as reinforcement learning agent on the site at different time steps. The reinforcement learning agent uses deep convolutional neural networks with Q-learning algorithm to approximate the Q-function and that uses experience replay. The neural network used by the learning agent is a time dependent encoded AI model and trained with Reinforcement learning by different methods to make an autonomous robots to learn how to execute set targets by way of reliable tracking of the environment for exhibiting a realistic behaviour by robots. The results of the study simulated on existing internet here on Earth show that the real individual behaviour on a distant planet can be achieved provided the interplanet internet is available as pathway communication.

*

Therefore, connected metaverse with different time-dependent encoded layers of virtual spaces along with deep learning models with learning agents could be of reality even in interplanet environment.

INTRODUCTION

Inter-planetary exploration, be it Lunar habitation, asteroid mining, Mars colonization or planetary science/mapping missions of the solar system, will increase demands for inter-planetary communications. The movement of people and material throughout the solar system will create the economic necessity for an information highway to move data throughout the solar system in support of inter-planetary exploration and exploitation. The communication capabilities of this interplanet information highway need to be designed to offer; 1) continuous data, 2) reliable communications, 3) high bandwidth and 4) accommodate data, voice and video.

The interplanetary Internet is a conceived computer network in space, consisting of a set of network nodes that can communicate with each other. These nodes are the planet's orbiters (satellites) and landers (e.g., robots), and the earth ground stations. For example, the orbiters collect the scientific data from the Landers on Mars through near-Mars communication links; transmit the data to Earth through direct links from the Mars orbiters to the Earth ground stations, and finally the data can be routed through Earth's internal internet. Interplanetary communication is greatly delayed by interplanetary distances, so a new set of protocols and technology that are tolerant to large delays and errors are required. The interplanetary Internet is a store and forward network of internets that is often disconnected, has a wireless backbone fraught with error-prone links and delays ranging from tens of minutes to even hours, even when there is a connection. In the core implementation of Interplanetary Internet, satellites orbiting a planet communicate to other planet's satellites. Simultaneously, these planets revolve around the Sun with long distances, and thus many challenges face the communications. The reasons and the resultant challenges are: The interplanetary communication is greatly delayed due to the interplanet distances and the motion of the planets. The interplanetary communication also suspends due to the solar conjunction, when the sun's radiation hinders the direct communication between the planets. As such, the communication characterizes lossy links and intermittent link connectivity.

*

The graph of participating nodes in a specific planet to a specific planet communication, keeps changing over time, due to the constant motion. The routes of the planet-to-planet communication are planned and scheduled rather than being fluctuating. The Interplanetary Internet design must address these challenges to operate successfully and achieve good communication with other planets.

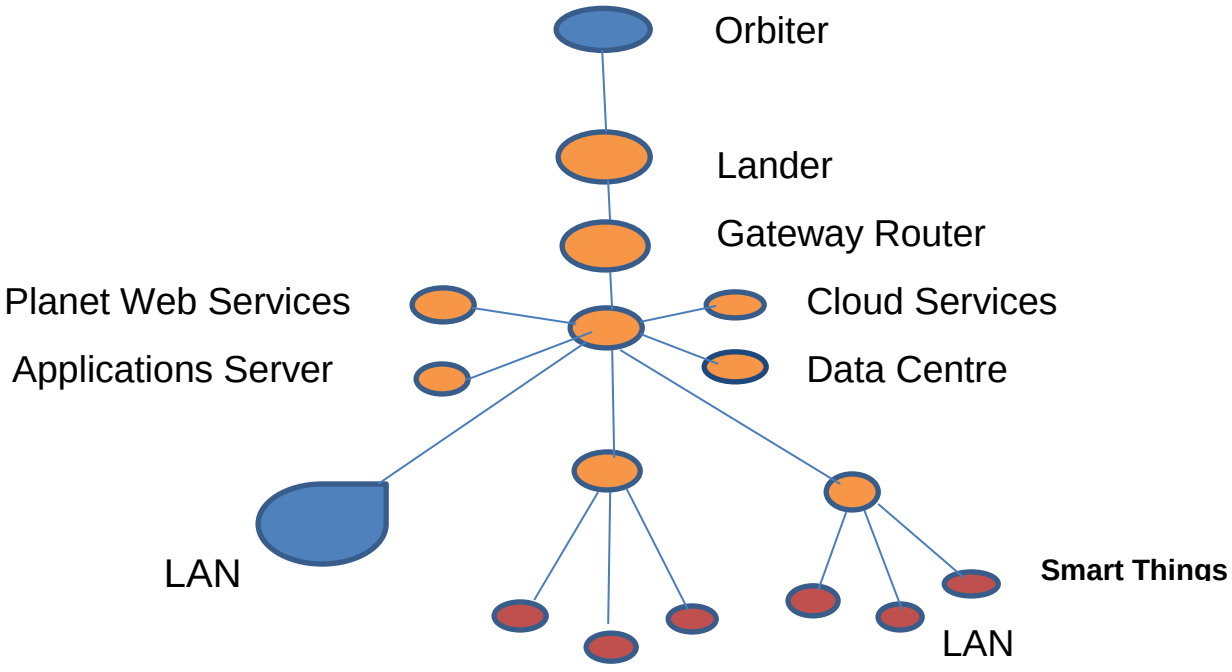
NETWORK ARCHITECTURE

A **Computer Network Architecture** is a design in which all computers in a computer network are organized. An architecture defines how the computers should get connected to get the maximum advantages of a computer network such as better response time, security, scalability, etc.

Network architecture refers to the way network devices and services are structured to serve the connectivity needs of client devices.

- Network devices typically include switches and routers.
- Types of services include DHCP and DNS.
- Client devices comprise end-user devices, servers, and smart things.

We give below the architecture of network on the planet Mars or the Earth's Moon is as shown in below figure:-



Local Area Network

An Internet is a "network of networks" in which routers move data among a multiplicity of networks with multiple admin. domains.

*

The main aim of networks is to connect remote endpoints with end-to-end principle and network should provide only those services that cannot be provided effectively by endpoints.

The interplanetary internet is a conceived networks of nodes and these nodes are space station, planet's orbiters (satellites), planet's landers, robots (drones, autonomous machines, etc.), earth ground stations and earth's internal internet.

METHODOLOGY

Outer space contains a vast amount of resources that offer virtually unlimited wealth to the humans that can access and use them for commercial purposes. One of the key technologies for harvesting these resources is robotic mining of minerals, metals, etc. The harsh environment and vast distances create challenges that are handled best by robotic machines working in collaboration with human explorers. Humans will visit outposts and mining camps as required for exploration, and scientific research, but a continuous presence is most likely to be provided by robotic mining machines that are remotely controlled by humans either from Earth or from local space habitat.

Future **Moon(or Mars)** bases will likely be constructed using resources mined from the surface of the Moon/Mars. The difficulty of maintaining a human workforce on the Moon(or Mars) and communications lag with Earth means that mining will need to be conducted using **collaborative robots** with a high degree of autonomy. Therefore, the utility of autonomous collaborative robotics(with thousands of robots in operation) towards addressing several major challenges in autonomous mining in the lunar(Martian) environment with lack of satellite communication systems, navigation in hazardous terrain, and delicate robot interactions to achieve effective collaboration between robots and long-lasting operation.

Collaborative Robotics

Robots can be shaped to perform specific tasks. Robots have been designed and shaped in such a way that they can walk, swim, push pellets, carry payloads, carry shoveling and work together in a group to aggregate debris scattered along the surface into neat piles or possibly, to build a space settlement. They can survive for long-time without recharge and heal themselves after any damage/confusion. The shape

*

of a robot's body, and its distribution of legs and structure are automatically designed in simulation to perform a specific task, using a process of trial and error.

The methodology is essentially fundamental for getting the space robots as autonomous as possible and the aim is to represent surroundings and their markings from robot in space. Therefore, we use feature extraction from the environment to update the position of the robot. Landmarks are the features that can easily be observed and distinguished from the environment and these are used to localize the robot.

The methodology primarily consists of following parts:-

1. Selecting and deciding on the landmarks (Materials, location, etc.).
2. Extracting landmarks from input of robot sensors/cameras at each time step.
3. Based on time step, get the current position of the robot on the basis of landmark data.
4. Carryout landmark data association with the location of the robot by matching with the landmarks data in the database.
5. Introducing learning agent (Reinforcement Learning) in the robot that uses deep neural network with Q-learning algorithm and that uses experience replay
6. The neural network used by the learning agent will be trained with reinforcement learning by using different methods.
7. Measuring the outcome with optimization steps

Single-agent Reinforcement Learning

Agents

An agent: “it is a computer system situated in some environment, and capable of autonomous action in this environment in order to meet its design objectives”, goals that it tries to reach. Indeed, we can distinguish between two key characteristics of an agent: its reactivity, its ability to perceive the environment and to respond to changes in it, and its proactivity, its ability to take initiatives and act towards its goal.

Reinforcement Learning

In machine learning, we distinguish between three types of learning: supervised, unsupervised and reinforcement learning. The distinction is usually made by the feedback the agent receives.

*

In between these two cases of supervised and unsupervised learning stands reinforcement learning where the feedback exists but does not indicate whether the action taken was the right one. After acting, the agent gets a feedback, an immediate reward that can be either positive or negative. It is up to the agent, and thus the learning algorithm, to use and interpret this reward. Reinforcement learning can be seen as a trial and error approach. As the agent is not explicitly told which action to take, it can only evaluate the actions with the rewards it received. For this evaluation to be efficient, the agent has to continually interact with the environment and adapt its strategy with regard to the rewards it gets.

Q-Learning

This value iteration algorithm uses explicitly the state transition probability function T and the reward function R of the MDP(Markov Decision Process) . However, it is usually assumed that the *model*, which consists of knowledge of T and R, is unknown. In this case, we distinguish between two approaches. *Model-based* algorithms attempt to learn the model and use the estimate of the model to compute an optimal policy while *model-free* methods focus on learning the state value function and use these estimates to get an optimal policy. Such methods are generally known as *temporal difference methods.*
The *Q-learning* algorithm is one of the most popular reinforcement learning techniques. It is a *model-free* value iteration algorithm.
We define the action-value function, or Q-function, $Q^{\pi} : S \times A \rightarrow \mathrm{R}$ as the expected return of a state-action pair given by the policy π . The optimal Q-function is then defined as $Q^{*}(s, a) = \max_{\pi} Q^{\pi}(s, a)$.

Deep Reinforcement Learning

Deep learning is defined as “a class of machine learning techniques that exploit many layers of non-linear information processing for supervised or unsupervised feature extraction and transformation, and for pattern analysis and classification”. Other definitions are given but we can observe the same two key concepts across all these definitions as such, complex neural networks fit in this definition.
Deep Reinforcement Learning refers to the usage of deep learning in a reinforcement learning setting.

Deep Q-Networks

The deep reinforcement learning field’s popularity started with the introduction of Deep Q-Networks (DQNs) (Mnih et al., 2015). DQNs are deep learning models that combine deep convolutional neural networks

*

with the Q-learning algorithm. Before this, using a nonlinear function such as a neural network as a function approximation for the Q-function was known to be unstable. An important contribution made by DQNs is the use of experience replay and a target network to stabilize the training of the Q action value function approximation with deep neural networks. Q-network thus refers to a neural network function approximation of the Q-function: $Q(s,a;\theta) \approx Q*(s,a)$ Where θ is the weights of the network.

Deep Q-Learning

The deep Q-learning algorithm uses experience replay. An agent's experience at a time step t is denoted by e_t and is a tuple (s_t,a_t,r_t,s_{t+1}) consisting of the current state s_t, the chosen action a_t, the reward r_t and the next state s_{t+1}. The experiences for all the time steps are stored in a replay memory, over many episodes. We then apply minibatches updates to samples of experiences.
Since the network serves as an approximator for the Q-function, each output neuron of this network corresponds to one valid action, and every action is mapped to an output neuron. Thus, after a feedforward pass of that network, the outputs are the estimated Q-values of the state-action pair defined by the input and the output's corresponding action.

Multi-Agent Systems

A multi-agent system, subsequently referred to as MAS, is a system composed of numerous agents and the environment in which they interact (Wooldridge, 2002).
A MAS is essentially a distributed approach (e.g. robots teams, traffic light networks, electricity grids, etc.). The complexity of multi-agent systems often means that we do not always know what action an agent should have taken. We can, however, determine when an action was good, whether it led to an optimal situation or just a better one. This knowledge should help the learning of the agents and corresponds to the reinforcement learning approach.

ARCHITECTURE

Agents – Attributes

We opt for the agents and they have the attributes: the sight and the goal. While the goal is chosen randomly when an agent arrives on the location, the sight is always fixed to the some value. The other noticeable fact is that our learning agents do not have a desired speed. We define the autonomous robots as entities whose primary concern is to avoid failure; they should consequently not exhibit any preference for

*

a certain speed as long as they are working safely. Furthermore, we add an attribute ϵ to these learning agents; this is their probability of choosing a random action at each time step.

Rewards

We need to define the reward function R; there are three different final states the agents can be in. First, they can reach their goal. Second, they can miss their goal. Finally, they can collapse. Hence, we define the following rewards:

- ρ_{ω} , the reward received by the agent when it reached its goal
- ρ_{ω^-} , the reward received by the agent when it failed to reach its goal
- ρ_f , the reward received by the agent when it collapsed

Since reaching the goal and collapsing are completely opposite outcomes, we define $\rho_{\omega} = - \rho_f$. Moreover, since missing the goal but not collapsing is preferable but is a less desirable outcome than reaching the goal, the value of this reward should be smaller, such that $\rho_f = -\rho_{\omega} < \rho_{\omega^-} < \rho_{\omega}$. The agent gets these final rewards at the time step that effectively ends its run on the site. For all the other previous time steps, it receives a default reward that is always equal to 0. However, the agent receives another reward ρ0, a small penalty, when the agent's speed is 0; our environment is a time bound activity and robots should not be still.

Agents as workmen

Given that we define learning agents the same way as the type of workers, we can seamlessly add them at the location. The only difference is how they will choose an action: by using their learning model, a neural network. We can therefore adapt the site's time step's algorithm to take the learning agent into account for the observation step. To decide what action it should take, the reinforcement learning agent uses a neural network to approximate the Q-function. Thus, at every time step t, the agent c observes its state $s_{c,t}$; this state is then processed in some way so that it can be passed to a neural network whose outputs correspond to all the possible actions. The values of these outputs are the estimated Q-values, $Q(s_{c,t}, a)$; as it is using a neural network θ, we denote the Q-function approximated with that network by Q(s,t;θ). The agent then uses an ϵ-greedy strategy to choose the action $a_{c,t}$.

The neural network used by the learning agent will be trained with reinforcement learning by using different methods.

Neural Network Models

Presently different neural network models are available that we will use to train our autonomous robots. These models define what information the learning agents use and how they are encoded as inputs to the

*

neural networks. Before we start with our model, we need to define the building structure; how these neural networks are used by the learning agents. We use a feedforward neural network whose outputs correspond to the possible actions. Our models define different ways of using information about the agent's current state. Thus, they either encode different information or encode the same information differently to produce the inputs.

Required Information

We start by defining the minimum amount of information that an autonomous robot should have. Consequently, the model that we design will possess these pieces of information. They are:

- The goal of the agent/robot
- What the robot sees, Materials & location
- The current location that the agent is in
- The current speed of the agent
- Real Simulation for Task Execution

Task Execution (Simulation)

- Generating actual materials(how materials arrive at the site)
- Robots arrive in the environment (speed and goal)
- Task Execution(Simulation Steps), is updated as the work process progresses in line with the simulation
- Task execution performance and to make a realistic system, we would like to see how well it performs and mirrors real world execution(Artificial Intelligence with Reinforcement learning)
- Implementation of Graphical Version of the Task Execution

Reddy's Encoding Model

The model is based on the idea as the robot presence at different time steps; we use information about the previous time step (the robots' presence represented by the observation matrix O). This time, the observation matrix of the previous time step t−1, denoted Ot−1, is not additional inputs, but it forms, along with the current observation matrix, a 3-dimensional matrix with time as the third dimension. We then pass this matrix through a 3-dimensional convolutional neural network . We also keep decreasing the number of inputs by including the learning agent itself in its observation matrix. Figure 1 illustrates this model that we call reddy's encoding model of time-step, as it encodes the robots' presence at different time steps. The learning agent is shown in red while materials in its field of view are in blue. All the input vectors are concatenated and passed to the network.

*

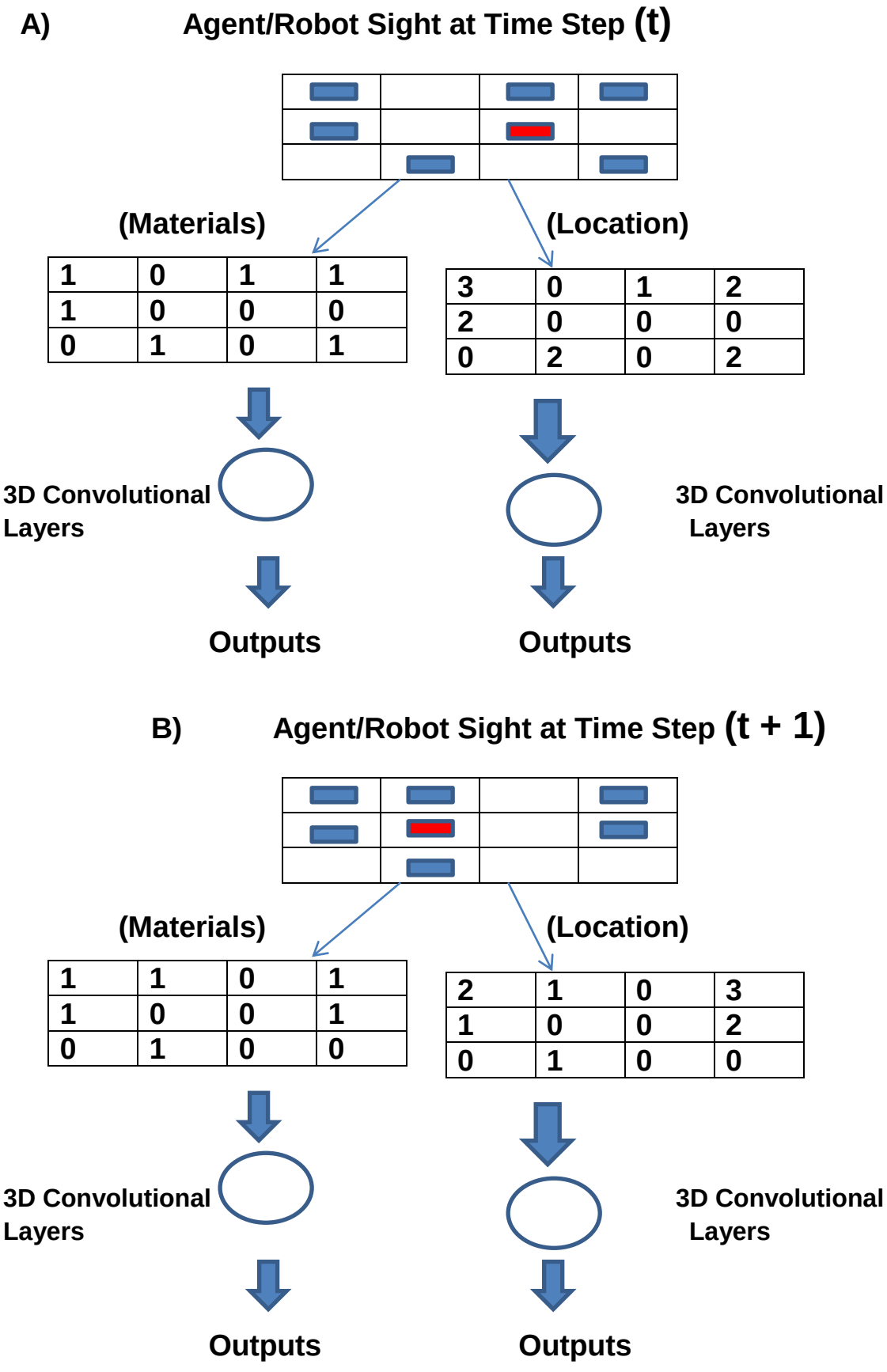


Fig 1. Model encoding of an agent/robot with a goal.

*

Our first step is the basic information regarding the robot vision and it's location. We define the model with each input as either 0 or 1 and information regarding the field of view of the agent/robot can only be 1 if there is something , or 0 if there is nothing. The agent/robot only knows whether there is a material to be used in some position and its location with respect to materials identified.

We have represented this model as matrix with encoded values with possible values for each of these attributes.

Deep Reinforcement Learning

Single-Agent Setting

We first train autonomous robots in a single-agent environment; there is only one learning agent on the location of site, other robots are our simulated workmen. The model was implemented in Python with TensorFlow as back-end.

Training - Training Scheme

Now that we have defined our neural network model and the work process, it is time to tackle our main problem: making an autonomous robot learn how to work.

The training environment set up for our learning agent is divided into episodes. One episode consists of a full run, or simulation, of the agent on the location.

First, as the agent could potentially stay still indefinitely, if it always chooses to stay at the speed zero, we need to guarantee that its run on the site will come to an end. We thus define a maximum number of steps T^{max}; if the agent's run on the location reaches this upper bound, we consider the run over and introduce another reward, denoted ρ_T, for this new outcome: the overtime outcome. Furthermore, the value of T^{max} must be chosen wisely. The bound must not be too low or too great. As the time required for an agent to leave the site depends on the size of the locational site, we need to define the overtime bound based on this value. We could then use twice that value, 2×X, but it means that the agent could potential stay still half of the time and still finish its work. Finally, we define arbitrarily T^{max} as 2×X −10; we want the agent to actually work more often than it is not moving.

Second, we perform an arbitrary number of location time step updates, usually 10, before adding the agent in the system. Then, we randomly initialize the attributes – speed v and location y – of the agent and add it on the locational site.

*

Finally, we use experience replay to train the network in an online fashion; at each time step, we sample a batch of experiences from the memory and train the network with it.
The training of the neural network –is done by the module with the weights updates. Also the model was composed with the stochastic gradient descent optimizer and the mean squared error loss function.
To train the network, module needs inputs for this network as well as the wanted target outputs of these inputs. Thus, we need to define these target outputs according to the chosen action. To do that, we define the target outputs as the outputs that we get from the network except for the chosen action. For this particular output, we set its target value to the newly estimated Q-value. That is, if the new state is final, the immediate reward r_t received; otherwise, the immediate reward and the discounted best possible future reward as estimated by the network. In short, when an agent in state s_k performs action a_k that yields the immediate reward r_k and arrives in the next state s_{k+1} , we define the target outputs y as:

$Q(s_k, a; \theta)$ if $a \neq a_k$

$r_k + \gamma \max_{aj} Q(s_{k+1}, a^j; \theta)$ if $a = a_k$

Where $Q(s_{k+1},a;\theta)$ is equal to 0 for every action a when s_{k+1} is final.
This whole process is formalized in below Algorithm.

Algorithm : Single-agent training algorithm Initialize replay memory D of length *M*

for *episode* = 1*, ..., N* **do**

Initialize location *L* according to fixed locational parameters

for *t* = 1*, ...,* 10 **do**

Perform one time step update of the location *L*

end

Initialize learning agent *c* with a fixed sight φ_c^+ and *s*

Choose randomly location *y* at which to add the agent, and speed *v* of the agent

Add the learning agent *c* on the site

for *t* = 1*, ..., T* ^{max} **do**

Perform one time step update of the location *L*

/* Experience replay */

*

Observe the state s_t, action a_t and reward r_t of the agent for that time step, and the next state s_{t+1}

Store experience (s_t, a_t, r_t, s_{t+1}) in D

if *replay memory* D *is full* **then**

Sample minibatch of size *B* from experience memory D

Initialize batch training buffer of size *B*

for *experience* (s_k, a_k, r_k, s_{k+1}) *in minibatch* **do**

Get network outputs **y** according to $Q(s_k, a; \theta)$

Set $y_{ak} = r_k$ if state s_{k+1} is final

Set $y_{a_k} = r_k + \gamma \max_{aj} Q(s_{k+1}, a^j; \theta)$ if state s_{k+1} is not final

Store (s_k, **y**) in batch training buffer

end

Train network θ against batch training buffer

end

end

end

Training Parameters

A training experiment is defined by numerous parameters; first, all the parameters regarding the learning agent and the training algorithm. These parameters are:

- φ_c^+ , the sight of the learning agent
- s, the agent's probability of choosing a random action
- *R*, the reward function; which corresponds to defining the different rewards ρ_ω, ρ_{ω^-} , ρ_f , ρ_0 and ρ_T
- μ_c, the neural network model used by the agent
- γ, the discount factor as defined for the MDP
- α, the learning rate for the neural network training
- *N* , the number of training episodes
- *M* , the size of the experience memory buffer

*

- *B*, the size of the batches of experiences

Moreover, the hidden structure of the neural network model μ_c can also be chosen; this includes the number of hidden layers, the number of neurons in these layers, their activation function, and the dropout rate δ_c (0 if we do not want to use dropout) to apply to each layer. Finally, if the chosen model is a CNN one, the structure of the convolutional networks can also be set: the number of convolutional layers with their stride and the number and size of the filters.
Learning – sampling batches of experiences to update the network's weights – starts once the experience memory buffer is full: after encountering *M* experiences.

Experiments

For our experiments, we train the model we presented earlier. Moreover, we also repeat the same experiments we trained our model with and We also considered different possible hidden structures for the neural networks.

Settings

For our experiments, most of the parameters are fixed. We present them in Table 1. The structure of the CNNs for the *time-step* model are also fixed with two convolutional layers, and the reward system of our learning agent is fixed, and the values are shown in Table 2.

RESULTS

Table 1: Single-agent training's fixed parameters

Learning parameters	
Number of episodes *N*	100
Batch size *B*	10
Experience memory size *M*	40
Learning rate α	0.01
Discount factor γ	0.9
Agent's sight φ^{+}_{c}	6
Agent's *s*	0.05

*

	Reward ρ	Value
Goal	ρ_{ω}	+1
Missed goal	$\rho_{\bar{\omega}}$	-0.15
Collapse	ρ_f	-1
No speed penalty	ρ_0	-0.01
Overtime	ρ_T	-0.4

Table 2: Single-agent training's reward

The system with different configurations for the hidden structures of the networks:

- 2 hidden layers: the first with 30 neurons and a *tanh* activation function; the second with 15 neurons and a *linear* activation fucntion. No dropout.
- 2 hidden layers: the first with 30 neurons and a *tanh* activation function; the second with 15 neurons and a *linear* activation fucntion. Dropout rate of 0.5.

The dropout rate of 0.5 has been chosen because it seems to be optimal for a wide range of networks.
The results for our CNN based model – *time-step* model – The networks that do not use dropout seem to learn well. The percentage of goal reached for the networks (without dropout) is high.
Although we only have partial results, we can make the following observations: the networks that do not use dropout seem to learn well, while the network using dropout does not; it either learns very slowly or just converges to very low level of goal reached.

CONCLUSION

The interplanetary computer network in space is a set of computer nodes that can communicate with each other. We proposed a network architecture with planet's orbiters, landers (robots, etc.), as well as the earth ground stations and linked through Earth's internal internet, and consisted of complex information routing through relay satellites to address direct planet-to-planet communication. As we know, the metaverse will be very different from the internet of today due to massive parallelism, three-dimensional (3D) virtual space and multiple real-world spaces like space mining, building space habitats, etc. This paper presents a time dependent observation matrix at different time steps along with autonomous learning agent in a planetary environment for layering the presence of robots and tracking the environment that use encoded model, AI and Reinforcement learning mimicking the real world execution by space robots. For this 3-dimensional matrix with time as

*

third dimension and AI- deep convolutional network with Q-learning algorithm that uses experience replay have been presented along to create a simulated realistic world and the results show that the real individual behaviour on a distant planet can be achieved provided the interplanet internet is available as pathway communication.

REFERENCE

1. **Poondru Prithvinath Reddy**: **“Metaverse in InterPlanet Internet: Modeling, Validation, and Experimental Implementation ”, Google Scholar**