simpread-MultiAgent Reinforcement learning for RoboCup Rescue Simulator

> 本文由 简悦 SimpRead 转码， 原文地址 towardsdatascience.com

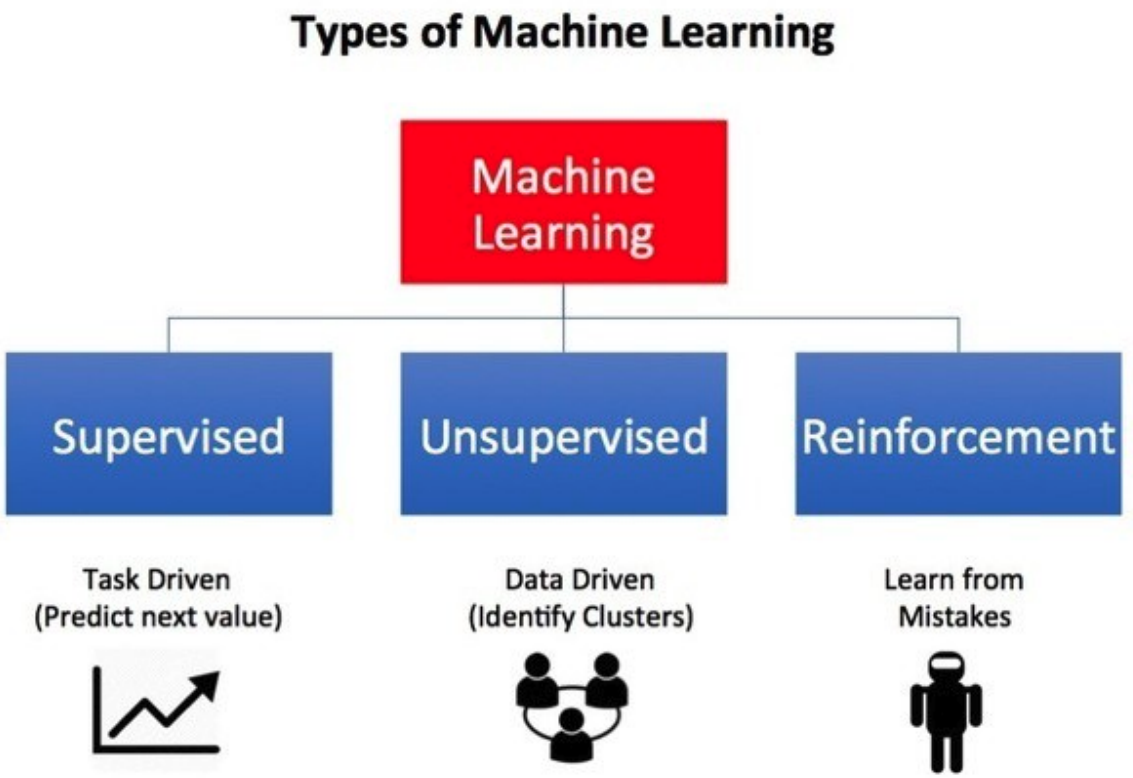
> MultiAgent, Reinforcement learning, RoboCup Rescue Simulator. Quite a lot of jargon used already in t.....

RoboCup Rescue Simulator

MultiAgent, Reinforcement learning, RoboCup Rescue Simulator. Quite a lot of jargon used already in the title. But don't worry. You will understand every bit of it after reading this article. I will start by building some background first about machine learning, types of machine learning techniques and then delve deep into the Reinforcement learning arena. This is where things will start to get slightly technical but I will try to keep it as simple as possible and provide examples wherever possible. Then I will explain how I applied Reinforcement learning to train a bunch of fire brigades to find buildings that are on fire and extinguish fire in those buildings. ALL BY THEIR OWN! So brace yourself for this exciting journey.

We are in an age where we can teach machines how to learn and some machines can even learn on their own. This magical phenomenon is called **Machine Learning**. But how do machines learn? They learn by finding patterns in similar data. Think of data as the

information you acquire from the world. The more data given to a machine, the “smarter” it gets.



Types of Machine learning algorithms

Broadly, there are three types: **Supervised learning, Unsupervised learning and Reinforcement learning.**

1. Supervised learning

The machine learns from labeled data. **Labeled data** consists of unlabeled data with a description, label or name of features in the data. E.g. In a labeled image dataset, an image is labeled as it is a cat’s photo and it’s a dog’s photo.

2. Unsupervised learning

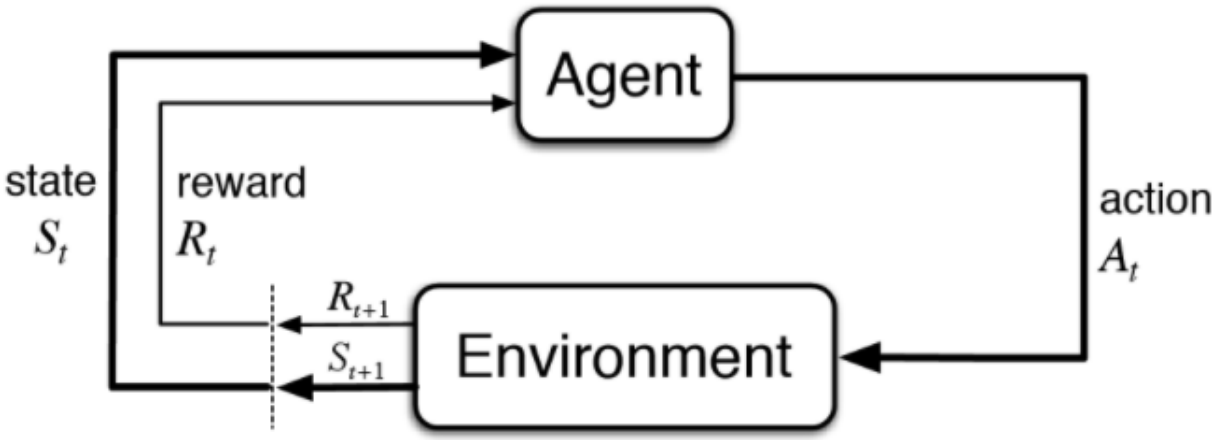
The machine learns from unlabeled data. **Unlabeled data** consists of data which is either taken from nature or created by human to explore the scientific patterns behind it. Some examples of unlabeled data might include photos, audio recordings, videos, news articles, tweets, x-rays, etc. The main concept is there is no explanation, label, tag, class or name for the features in data.

3. Reinforcement Learning

It is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences.

Though both supervised and reinforcement learning use mapping between input and output, unlike supervised learning where the feedback provided to the agent is a **correct set of actions** for performing a task, reinforcement learning uses **rewards and punishments** as signals for positive and negative behavior.

As compared to unsupervised learning, reinforcement learning is different in terms of goals. While the goal in unsupervised learning is to find similarities and differences between data points, in the case of reinforcement learning the goal is to find a suitable action model that would maximize the **total cumulative reward** of the agent. The figure below illustrates the **action-reward feedback loop** of a generic RL model.

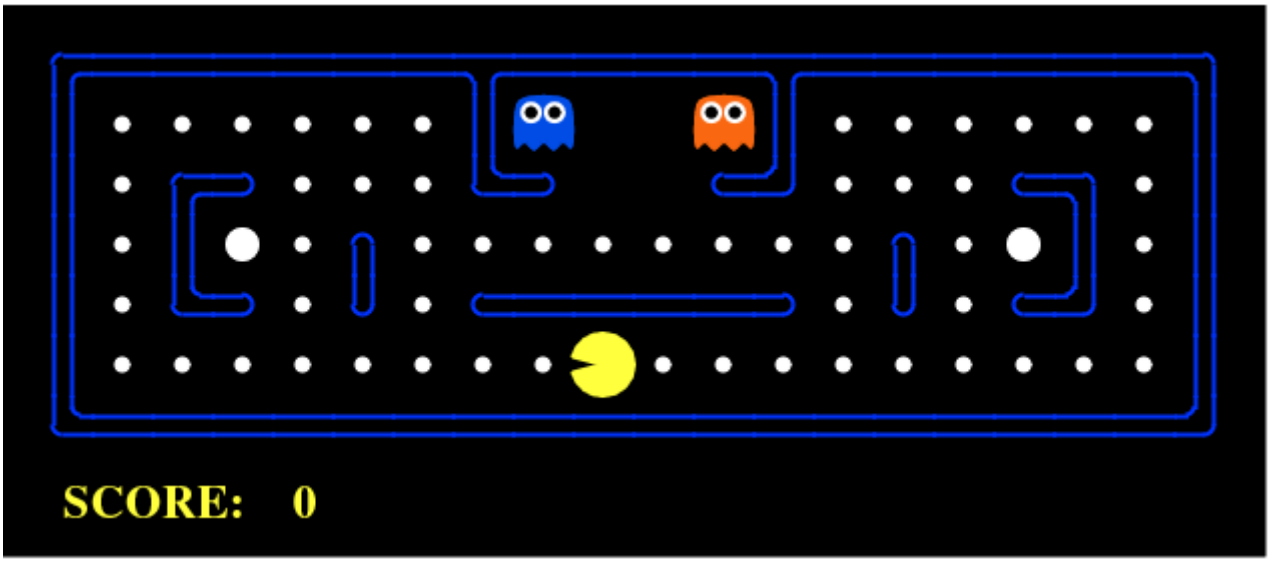


Abstract depiction of the Reinforcement learning model

Some key terms that describe the basic elements of an RL problem are:

1. **Environment** — Physical world in which the agent operates
2. **State** — Current situation of the agent
3. **Reward** — Feedback from the environment
4. **Policy** — Method to map an agent's state to actions
5. **Value** — Future reward that an agent would receive by taking an action in a particular state

An RL problem can be best explained through games. Let's take the game of PacMan where the goal of the agent(PacMan) is to eat the food in the grid while avoiding the ghosts on its way. In this case, the grid world is an interactive environment for the agent where it acts. The agent receives a reward for eating food and punishment if it gets killed by the ghost (loses the game). The states are the location of the agent in the grid world and the total cumulative reward is the agent winning the game.



Example of Reinforcement learning applied to PacMan

In order to build an optimal policy, the agent faces the dilemma of exploring new states while maximizing its overall reward at the same time. This is called **Exploration vs Exploitation** trade-off. To balance both, the best overall strategy may involve short term sacrifices. Therefore, the agent should collect enough information to make the best overall decision in the future.

Markov Decision Processes**(MDPs)** are mathematical frameworks to describe an environment in RL and almost all RL problems can be formulated using MDPs. An MDP consists of a set of finite environment states S, a set of possible actions A(s) in each state, a real-valued reward function R(s) and a transition model P(s', s | a). However, real-world environments are more likely to lack any prior knowledge of environment dynamics. Model-free RL methods come handy in such cases.

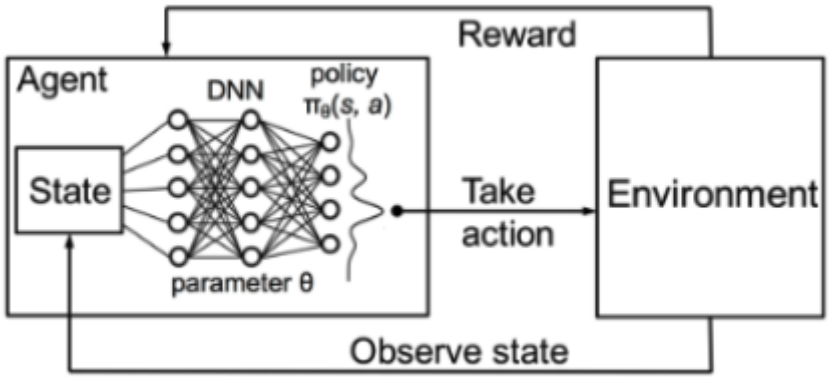
Q-learning is a commonly used model-free approach that can be used for building a self-playing PacMan agent. It revolves around the notion of updating Q values which denotes the value of performing action *a* in state *s*. The following value update rule is the core of the Q-learning algorithm.

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

Q-learning algorithm

Policy Gradient is another popular model-free RL method but it doesn't calculate Q-value but instead uses a policy. Policy learns a mapping from every state to action, and its objective is to find which actions lead to higher rewards and increase their probability. The policy gradient observes the environment and acts in it and keeps updating its policy based on the rewards it receives. After multiple iterations, policy converges to a maximum value. Drawbacks of the policy gradient method can be that it sometimes can get stuck in local maximum and will not be able to reach the global maximum.

In situations where there are a large state and action space, it is not feasible to learn the Q-values or policy for each state and action pair. This is when **Deep Reinforcement learning (DRL)** is used where neural networks are utilized to model the components of RL.



Deep Reinforcement learning architecture

We consider two DRL model-free based algorithms: Deep Q-Learning and ProximalPolicy Optimization mainly because DQN showed great results on Atari games and helped the team achieve human-level performance whereas using PPO, OpenAI's DOTA 2 team was able to beat 99.4 percent of players in a public match.

Deep Q-Network (DQN)

DQN is a combination of Q-Learning and deep neural networks. DQN addresses the in-stabilities caused by using non-linear approximator to represent the Q-value by using two insights: experience replay and the target network.

Experience replay: For instance, we put the last million transitions (or video frames) into a buffer and sample a mini-batch of samples of size 32 from this buffer to train the deep network. This forms an input dataset that is stable enough for training. As we randomly sample from the replay buffer, the data is more independent of each other and closer to i.i.d.

Target network: We create two deep networks θ- and θ. We use the first one to retrieve Q values while the second one includes all updates in the training. After say 100,000 updates, we synchronize θ- with θ. The purpose is to fix the Q-value targets temporarily so we don't have a moving target to chase. In addition, parameter changes do not impact θ- immediately and therefore even the input may not be 100% i.i.d., it will not incorrectly magnify its effect as mentioned before.

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}\left(\underbrace{r + \gamma \max_{a'} Q(s', a'; \theta_i^-)}_{\text{target}} - Q(s, a; \theta_i)\right)^2$$

With both experience replay and the target network, we have a more stable input and output to train the network and behave more like supervised training. To read more about how DQN works, read this amazing article by Jonathan Hui.

Proximal Policy Optimization (PPO)

PPO is a type of policy gradient method which performs comparably or better than state-of-the-art approaches while being much simpler to implement and tune. Careful tuning of the step size is required for achieving good results with policy gradient algorithms. Moreover, most policy gradient methods perform one gradient update per sampled trajectory and have high sample complexity. PPO algorithm solves both these problems. It uses a surrogate objective which is maximized while penalizing large changes to the policy. They defined a likelihood ratio

$$l_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}$$

PPO then optimizes the objective:

$$L^{CLIP} = \hat{E}_t[min(l_t(\theta)\hat{A}_t, clip(l_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

where A_t is the generalized advantage estimate and clip (l_t(θ),1− epsilon ,1 + epsilon) clips l_t(θ) in the interval [1 - epsilon, 1 + epsilon]. The algorithm alternates between sampling multiple trajectories from the policy and performing several epochs of SGD on the sampled dataset to optimize this surrogate objective. Since the state value function is also simultaneously approximated, the error for the value function approximation is also added to the surrogate objective to compute the complete objective function. To read more about how PPO works, read this amazing article by Jonathan Hui.

Now that the theory is in place, let's focus on some applications. On 17th January 1995, an earthquake hit the Japanese port city, Kobe, with a magnitude of 6.8 on the Richter scale causing mass destruction taking approximately 7000 lives. On April 30th 1999, the RoboCup Rescue league was founded in order to promote international research towards the development of better rescue capabilities. Over the years many teams have participated in the annual RoboCup Rescue league which consists of three competitions: robotics competition, virtual robot competition, and the simulated agent competition. We will be focusing on the simulated agent competition known as RoboCup Rescue Simulation (RCRS) which is a perfect example of multiagent systems. We will be applying reinforcement learning to RCRS in order to train these agents to accomplish their respective tasks.

RoboCup Rescue Simulation Environment

Below are the tasks that the agents are assigned:

- **Fire Brigades**: Extinguish Fire
- **Police Officers**: Remove debris
- **Ambulance**: Carry civilians to safe refuge

We will be focusing on fire brigades for this article and train them to learn by themselves to extinguish the fire in the city.

Reinforcement learning for RoboCup Rescue Simulator (RCRS)

As we discussed above, to formulate any RL model, we need to define the state space, action space and reward function. Lets discuss these attributes for RCRS.

State Space: State space has three parts. The first part is the building information which consists of the temperature and fieriness. Note that, fieriness is a parameter to measure the degree. of fire in a building. The second part is the agent information which gives the location((X, Y) coordinates), water in the fire tanks and the health points of the fire brigade at each timestep. The third part is the busy/idle information which is a binary variable. Fire brigades receive a building id at each timestep as their target location. But it sometimes takes more than one timestep for them to reach the building. In the meanwhile, actions are being sent continuously. Hence, fire brigades have to ignore the actions until the time they visit the building they have been told to visit in the previous timestep. This information is passed over as state information which will be highly valuable for our algorithm to perform better. Whenever the actions that are sent by the algorithm are used in the simulator (busy), 1 is sent back as the state information otherwise 0 is sent (idle).

State	Parameter	Range
Building Information	Temperature of Building	0-100
	Fieryness of Building	0-10
Agent Information	(X, Y) Coordinates	0-10000
	Water Level	0-15000
	Health Points	0-10000
Busy/idle Information	Binary variable	0/1

Range for state-space parameters

Action space: The only action available to our agent is to move to the building which is on fire and therefore the action space consists of the ID's of the buildings. Note that extinguishing the fire and refilling water are default characteristics of our agent i.e. whenever our agent is near a building on fire, it will try to extinguish it and whenever it is out of the water, it will move to the refuge to refill the tank. Therefore these actions are not included in the action space.

Reward function: Since the ultimate goal of the fire brigades is to extinguish the fire as quickly as possible, we created a reward function that awards the agents higher rewards for keeping the fire to a minimum and penalize them if the fire increases. Fieryness is one parameter that measures the degree of burn in the building and hence keeping the overall fieriness value to minimum results in a higher cumulative reward.

Fieryness Value	Reward Value
0-2	+10
3-5	-5
6-10	-10

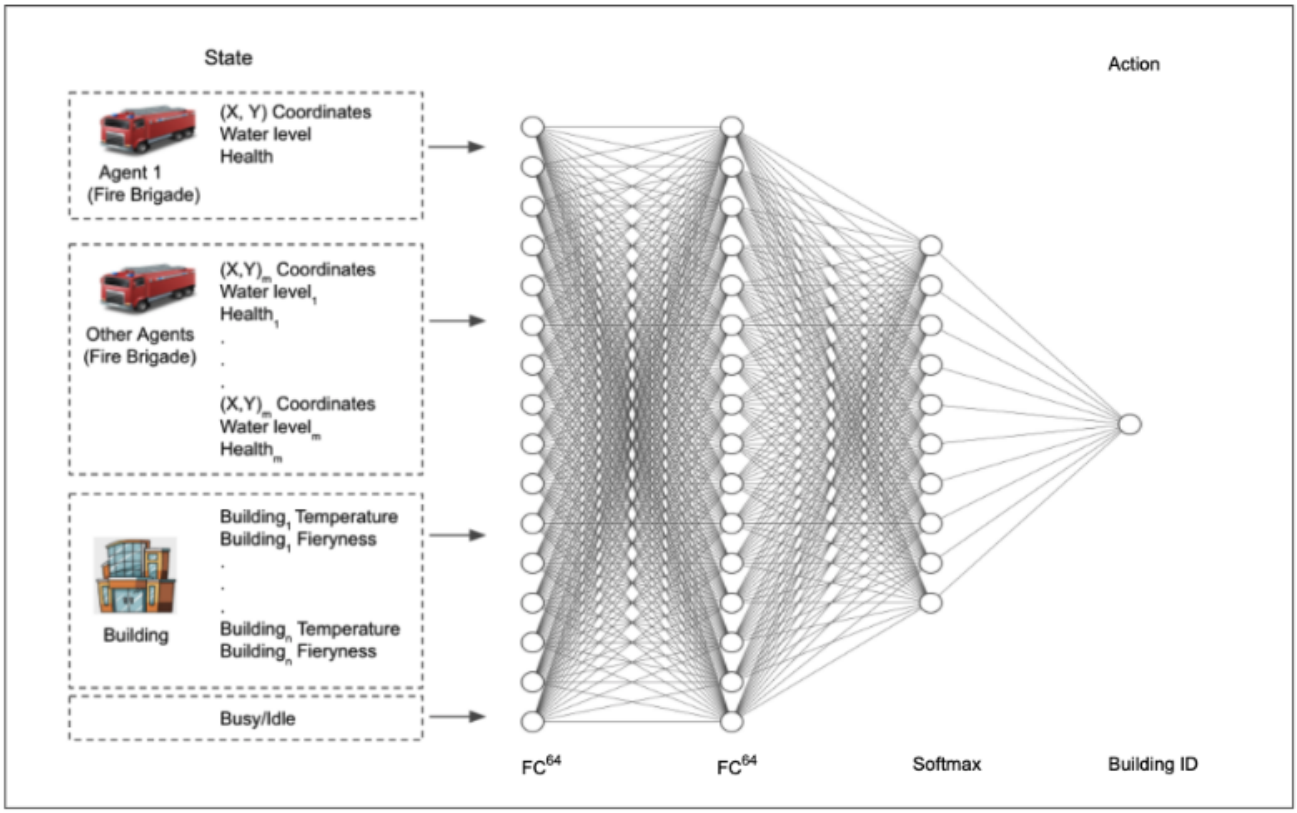
Reward Calculation

Fieryness Value	Severity
0-2	Slightly burned
3-5	Moderately burned
6-8	Critically burned
9-10	Totally burned

Fieryness severity according to fieryness value

Model Architecture

We use DQN and PPO architecture with a few modifications. We do not have the set of convolutional layers since the input to the neural networks is not an image. The input to the network is state representation and there is a separate output unit for each possible action. Stable baselines have the option of using different architectures where we can change the number of hidden layers, use LSTM, use CNN or combine CNN and LSTM. We tried different architectures while training the model. As an example of one of the architectures we tried, refer to the below figure.



Model Architecture

Hyperparameter search

The table shows the different hyperparameters I tried to train the model.

Parameter	Values
Discount factor (γ)	.99, .993, .997, .999
n-step	32, 64, 128, 256
Entropy Coefficient	0.01
Learning rate	log-uniform (1e7 $\rightarrow$ 1e3)
Value Function Coefficient	0.5
Gradient Clipping value	0.5
Number of training minibatches	4
Clip Range	0.2
Optimizer	Adam

PPO hyperparameters

Parameter	Values
Discount factor (γ)	.99, .993, .997, .999
Learning rate	log-uniform (1e7 $\rightarrow$ 1e3)
Buffer size	25000, 50000, 100000
Exploration Fraction	0.1
Batch size	32, 64, 128, 256
Learning starts	1000, 2000
Target Network update frequency	500
Optimizer	Adam

DQN hyperparameters

Too much theory! Let's get down to business end. What are the results?

Below are some of the simulations of trained agents on DQN and PPO. These videos have been taken at particular intervals i.e. after 5 episodes, 150 episodes and 250 episodes. Each episode is 100 timesteps long.

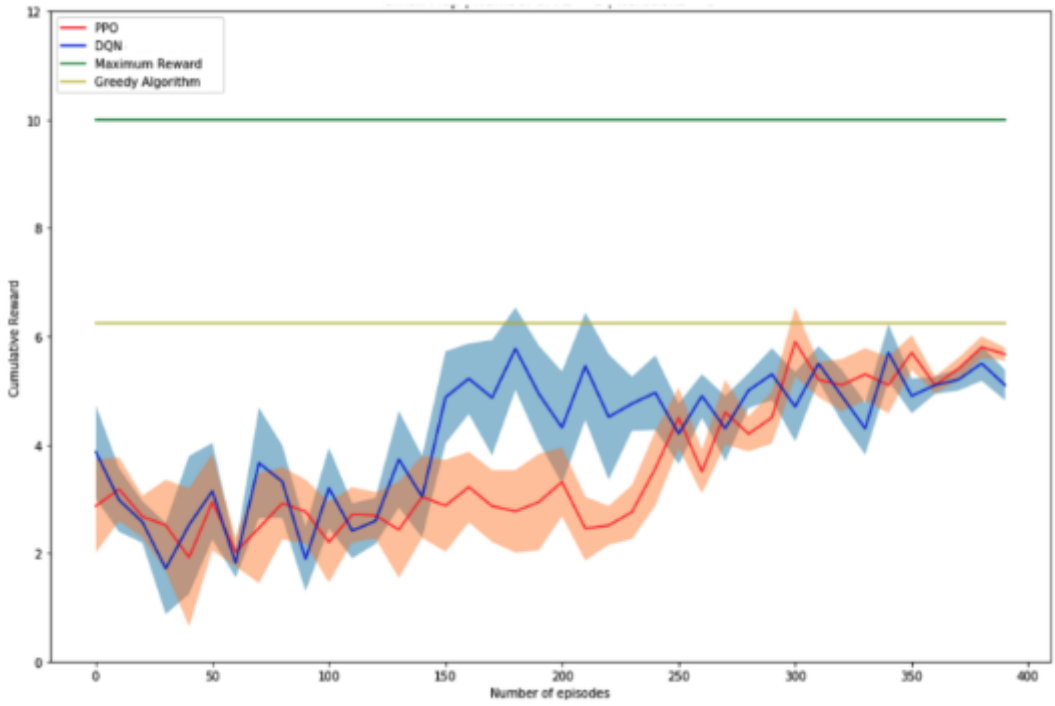
Here the red moving dot represents the fire brigade and to the bottom right corner is the refuge where fire brigades are refilling their water tanks.

Different color shades represent the intensity of the fire in those buildings:

- *Light yellow color*: Fire starting in the building
- *Orange color*: Fire in the building intensifies
- *Red color*: Building is burning severely
- *Black color*: Building is burnt out
- *Blue color:* Fire is extinguished in the building by the fire brigade

- *Grey color*: Building is unburnt

Here is a visualization of how the simulator looks like when it is untrained and how it starts learning to finally extinguish all the fire in the city.



The learning curve for both the algorithms

As we can see from the above curve and the gif's, both the algorithm were successful in learning on this simulator and were able to accomplish their tasks successfully. From the learning curve, we can also see that DQN performed better than PPO in this case and was able to learn quickly. Still the results were not better than the greedy algorithm (where all the agents would extinguish the building with the highest temperature). Currently, I am working on training these agents on a map with a higher number of buildings and will be sharing the results soon.

I would like to thank my mentors Peter Stone, Garrett Warnell and Tzu-Chiu for giving me the opportunity to work on this project.

Here's a link to my GitHub Repo.