A Study on Deep Reinforcement Learning for Dynamic Spectrum Access

Trevor Gordon
M.S Electrical Engineering
Columbia University
tjg2148@columbia.edu

***Abstract*—The number of devices communicating using the RF spectrum continues to increase. Historically, there have been centralized rigid rules for what devices can access specific frequency bands at any given time. As the number of devices increases, we need to find ways to share the frequency spectrum more efficiently. Deep Reinforcement Learning (DRL) is a promising framework for autonomous agents to learn usage patterns in the frequency spectrum and dynamically adapt to changing environments. In this report I provide an overview of recent literature using reinforcement learning to solve this dynamic spectrum access problem. I implement several ideas from existing papers and test new methods to achieve optimal frequency usage and fairness in a completely decentralized manner. I provide an in depth analysis of several experiments where agents learn to de-conflict their spectrum usage. Lastly, I provide a look into the decision making of agents as they learn in these scenarios by showing their changing value functions.**

***Index Terms*—Deep Reinforcement Learning, Dynamic Spectrum Access.**

I. Introduction

The number of devices using the frequency spectrum continues to increase. Typically the available spectrum has been divided up using rigid rules. As the frequency spectrum is a finite resource, we need find better ways of sharing. Sharing can be done by users sensing the available spectrum and transmitting where there is availability. However, it can be challenging to determine how to sense the spectrum and when to transmit. This scenario can be dynamic with new users joining or leaving. Further, some users may have very predictable usage patterns where others might be sporadic. Deep Reinforcement Learning (DRL) is a promising framework for autonomous agents to learn usage patterns in the frequency spectrum and dynamically adapt to changing environments.

II. Literature Review

A. Reinforcement Learning

Reinforcement learning is a type of machine learning that is concerned with agents that need to learn how best to take actions in their environment. At each time step agents receive an observation $\mathcal{O}$ from their environment, they choose an action $\mathcal{A}$ and receive a reward $\mathcal{R}$. In general the observation is derived from but not always equal to the agents state S. Typically there is a Markovian assumption that an agent's state captures all useful information. It doesn't matter how an agent got to a specific state for making decisions about the future. The mapping from observation to action is called the agent's policy $\vec{\pi}$. Reinforcement learning is broken down into model-based and model-free reinforcement learning. In model based RL, agents know the environment transition function $P(S_{t+1} \mid S_t, A_t)$. In model-free RL, they do not. In model free RL, agents learn how to act by keeping track of the value of a state V or the value of a state action pair Q which are the expectation of the total future γ discounted rewards. Q learning is a type of model free RL where agents keep a value table and choose actions based on the highest value state action pair available at that time. Agents start with randomly initialized values. Agents update the value table using the Bellman Equation shown below

$$\underbrace{\text{New}Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \underset{\text{Learning Rate}}{\alpha}[\underbrace{R(s,a)}_{\text{Reward}} + \underset{\text{Discount rate}}{\gamma}\overbrace{\max Q'(s',a')}^{\substack{\text{Maximum predicted reward, given} \\ \text{new state and all possible actions}}} - Q(s,a)]$$

It is common to use an exploration strategy like the epsilon greedy where agents choose a random action some percentage of the time so as to explore more states.

As described above, Q-Learning using a lookup table for the value of states. As the size of the state and action space increases it becomes impractical to keep track of so many values. The lookup table can be generalized as a function that takes in a state and action and returns a value. Deep Q Learning (DQN) is simply Q learning with a value function approximator realized as a Neural Network. However, there are some implementation details that are necessary for DQN to work. To train deep neural networks, it is typically required that training data is independent and evenly distributed. But, in reinforcement learning an agent's current state S_t and previous state S_{t-1} are highly correlated. To mitigate this issue it is common in DQN to store tuples of experience (State S_t, Action A_t, Reward R, Next State S_{t+1}, and whether the episode was finished) into a memory buffer. The agent then trains on random batches from it's memory buffer periodically. Like above, updates are made using the bellman equation. By sampling from the memory buffer we can get data is isn't so correlated. But, to perform the bellman update, we need the value of the observed next state $\max Q'(s', a')$ to update the value of the $Q(s, a)$. This estimate comes from the neural

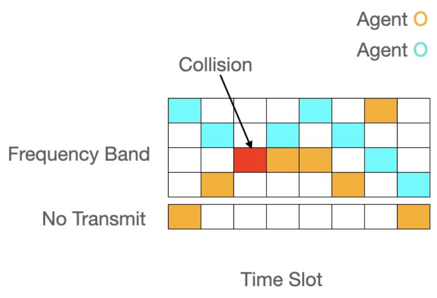

Figure 1. An example showing the dynamic spectrum access problem. The x axis here represents a discrete time interval. Filled in color squares represent the chosen action of each agent at that time step. Red squares represent collisions. The y axis represent the available actions to each agent. The bottom show corresponds with agents choosing not to act. There can be no collision if an agent chooses not to act.

network itself. To introduce stability, two copies of the Q network are kept. We grab the value of the next state from a *target network* which has weights updated only periodically from the main *q network* which is trained.

Further, another modification is the use of a Dueling DQN network that estimates the value $V(S)$ of the state without considering the actions first. It also separately estimates a vector for the additive advantage each action gives to the value. These are then added together to get the $Q(S, A)$. This improves learning by being able to back propagate information about one state action pair to the others for that state.

B. Multi-Agent Reinforcement Learning

In normal machine learning there is only a single agent making decisions which affect it's environment. The problem becomes more complex in multi-agent reinforcement learning (MARL) where agent's actions not only affect their environment but they also affect future actions of other agents. In this case we typically can not make the Markovian assumption. Furthermore, situations where agents need to cooperate to receive the highest rewards but may be incentivized to defect.

C. Dynamic Spectrum Access

This paper considers a dynamic spectrum access problem where multiple agents are attempting to transmit data over a limited number of frequency channels. At each discrete time step, agents have the choice of attempting to transmit in any of N_f frequency bands or not transmitting. If agents choose to transmit on the same band there will be a collision. If a single agent transmits on a band, there is a successful transmission. The goal is to maximize the number of successful transmissions. In this paper we consider fairness as well and so there is also a second goal to minimize the distance between the number of successful transmissions of all agents. It is assumed that agents always have packets to transmit.

D. Reinforcement Learning for Dynamic Spectrum Access

Deep reinforcement learning is an attractive solution to the problem of uncoordinated spectrum access because it serves as a framework whereby agents can learn in an online fashion. Agents can start by randomly transmitting and observing what works. While there are some safety critical frequency bands, in general it is much less risky for agents to make mistakes and learn in this environment compared to something like a self driving car. We expect there to be hybrid situations where we have some learning agents and some legacy users who follow simple patterns or follow rigid rules when transmitting. We expect there to be patterns that autonomous agents can recognize and exploit.

There are many different hyper-parameters that can be changed when considering this problem. I give an overview of some of some of the choices that need to be made when setting up experiments in this area.

- **The Observations**: At the very least, agents are able to observe their own actions and whether they had a successful transmission. At most, agents can view the actions of all other agents
- **Rewards**: Should agents only receive +1 for successful transmissions? What about -1 for collisions?
- **Reinforcement Learning Algorithm**: What structure is used so that agents can make decisions and learn? DQN? DDQN? Actor Critic Methods?
- **Agent Scenarios**: Are all agents the same type of agent or are there some primary users.
- **Action Space**: Most basic agents can choose which frequency band or not to transmit. In future scenarios, agents could decide their transmission power etc.

The authors in [2] consider a multi agent RL partially observable scenario. They use a deep neural network that can aggregate observations over time to estimate each agents true state. They use a Dueling DQN network as described above. A diagram of their network architecture is shown in Figure 2 . The authors consider a scenario with centralized training where by all users have a communication link with a main hub. Agents send their experience to the hub which trains using all data periodically. The authors re-train whenever the current experience doesn't reflect the current dynamics in the environment. They show that agents can coordinate using this approach and learn to maximize channel utilization. They consider three different reward strategies.

1) **Competitive reward maximization** scenario where agents aim to increase their own successful transmission.
2) **Cooperative reward maximization** where agents aim to maximize the total sum of all agents rewards.
3) **Global utility** that considers fairness where agent rewards are normalized by the sum of successful transmissions by that user up until that point.

They find that the cooperative reward results in single agents taking the whole spectrum while some users never transmit. They found that the proportional fairness and competitive reward maximization work under their conditions. However, they mention that the competitive reward maximization only works under a centralized training scenario.

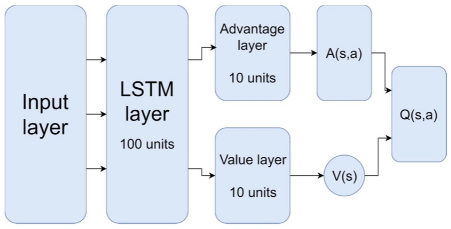

Figure 2. The DQN network structure used by citation needed. This image is taken directly from their paper.

Xu et. al. [4] also consider a deep recurrent network for Q learning realized as an Long Short Term Memory (LSTM) network. However, they also consider multiple different user types. They consider primary users who transmit based on simple rules. They also consider time changing situations whereby a second agent is added some time into the simulation.

Lastly, the authors in [5] consider an Actor Critic based RL solution. For the DQN models, the agent has a value function neural network and actions are taken in a greedy fashion. For an Actor critic method, the agent has a network for determining the value of state and a separate network for choosing which action is should take.

III. Experiment Setup

I setup my experiments primarily using the network used by [2] and shown in Figure 2.

At every time step, each autonomous agent decides between N + 1 actions where N is the number of frequency bands and the +1 corresponds with choosing not to transmit. It is assumed that all agents always have packets to transmit. If only one agent choosing to transmit on a specific band, there is a successful transmission. If two agents choose to transmit on the same bands, there is a collision. I focus on the following two goals:

- Maximize the number of successful transmissions
- Minimize the difference between successful transmissions of all agents

Unlike [2] I do not employ a centralized training scenario. I consider scenarios where agents learn based only on their own observations.

Below I discuss different variations of observations and rewards that correspond with possibilities in my software repository.

The Observation Type

Complete observability would mean that agents would have a history of all agents and their chosen actions. This would require some ledger and ability for all agents to read it. More realistically, I consider the following scenarios:

1) Observation type=Own Actions: Observation type where each agent has the least amount of information from the environment. Each agent can see a one hot encoded vector of it's own chosen actions and whether it got a reward. 0 for no transmission. 1 for successful transmissions. -1 for a collision The example below shows a slice along the agent dimension for 1 agent, a temporal length of 5, and 2 frequency bands.

		t	t-1	t-2	t-3	t-4
Action:	Freq Band 1	0	0	0	1	0
Action:	Freq Band 0	0	1	1	0	0
Action:	No Transmit	1	0	0	0	1
Success:		0	1	-1	-1	0

2) Observation type=Channel Status: Each agent can see whether there is communication or not on every channel. Channel is 0 for available and 1 for busy. Agents can not distinguish between successful transmissions or collisions on a frequency band. The example below shows a slice along the agent dimension for 1 agent, a temporal length of 5, and 2 frequency bands.

		t	t-1	t-2	t-3	t-4
Band Status:	Freq Band 1	0	1	0	1	1
Band Status:	Freq Band 0	0	1	1	0	0

3) Observation type=Aggregate: Includes the concatenation of all info from Own Actions and Channel Status. The example below shows a slice along the agent dimension for 1 agent, a temporal length of 5, and 2 frequency bands.

		t	t-1	t-2	t-3	t-4
Band Status:	Freq Band 1	0	1	0	1	1
Band Status:	Freq Band 0	0	1	1	0	0
Action:	Freq Band 1	0	0	0	1	0
Action:	Freq Band 0	0	1	1	0	0
Action:	No Transmit	1	0	0	0	1
Success:		0	1	-1	-1	0

The Reward Type

- *transmission1*: Agents receive +1 for successful transmissions and 0 otherwise.
- *collisionpenality1*: Agents receive +1 for successful transmissions, 0 for no transmit, and -1 for collisions.
- *collisionpenality2*: Agents receive +2 for successful transmissions, 0 for no transmit, and -1 for collisions.
- *centralized*: All agents receive the sum of all agent rewards.
- *transmission normalized*: Original rewards taken from *collisionpenality2* then scaled by the following equations. Rewards are divided by the number of successful transmissions in the reward history buffer. Intended to make transmissions more valuable to those who haven't gotten them. And to make transmissions less valuable to those who have been sending. Collisions are divided by the number of no transmits in the reward hist buffer. Intended to make collisions less costly to those who haven't been

transmitting. And to make collisions more costly to those who have been successful .

$$R = \begin{Bmatrix} R * RM+ & \text{if successful transmission} \\ R * RM- & \text{else collision} \end{Bmatrix}$$

$$RM+ = \frac{NT/RH}{1+NS/RH}$$

$$RM- = \frac{NS/RH}{1+NT/RH}$$

where:

RM+ Multiplier for successes

RM- Multiplier for collisions

RH Length of the reward history

ST Number of successful transmissions in the reward history

NT Number of times agent didn't transmit in the reward history

Other Experiment Hyper Parameters

- *Reward History Length*: When using transmission normalized this is the number of recent time steps used to calculate number of successes, collisions and no transmits
- *RL Model Type*: Most experiments are run with the DQN model. Also possible to use Actor Critic RL model.
- *Learning Rate*: Learning rate for training the network
- *Epsilon Start*: The start probability of taking a random explore action
- *Epsilon Decay*: The decay amount for the start epsilon
- *Agents Shared Memory*: Whether or not agents write their experience to and train from a shared memory buffer
- *Memory Buffer Size*: The size of the experience replay. Decrease the memory buffer size for faster changing environment dynamics
- *Episode Length*: Total number of time steps used for training
- *Agent Homogeneity*: Describes whether all agents are of the same type or whether there's a single periodic primary user agent

IV. Software Overview

All code is available at https://github.com/Trevor16gordon/deep-rl-spectrum-access. I used Python and Tensorflow. I write my own custom OpenAI gym environment to represent the MARL environment with configurable reward and observation models. A summary of the files is given below:

- **environment.py**: A custom openai gym environment to represent the multi agent dynamic spectrum access problem. This file contains the logic for keeping track of agent's action and return rewards and observations to the agents. This environment can be configured for different types of reward and observation situations.
- **agent.py**: Contains different types of agents with their own policies (learned or static) for determining how they should act in their environment.
- **model.py**: Contains the tensorflow deep learning models
- **serve_dash_plotting.py**: This file is used for interactive visualization of a training run. Instructions for using are in the visualize section of the repository README.
- **run_all_experiments.py**: This file contains calls to run the specific experiments below with correct hyper parameters
- **utils.py**: This file contains utility functions mostly for reshaping array and preparing data to save

V. Results

In this results section I present four experiments to showcase the ability of agents to learn in different scenarios. The summary of hyper parameters used are listed in table VII.

A. One agent learning to coordinate

Before testing learning agents in a MARL scenario, we want to see whether one learning agent can discover simple patterns in the spectrum. In this scenario we have one primary user that periodically transmits and one learning agent using the architecture described above. To simplify the learning environment we first start in a simple single-agent environment. In this scenario, there is one primary agent that periodically transmits for some period, then stops transmitting. There is a single learning agent that needs to learn when it should transmit. The results are shown in Figure 3. In the bottom row of plots I provide the agents value of choosing each action for those particular time step. We can see that in the beginning, the agent slightly favors not transmitting as it's bottom row is slightly darker. After a short amount of time through random exploration the agent finds that it can receive higher rewards by transmitting. At this time it hasn't learned to avoid collisions yet and so the agent always choosing to transmit. A short amount of time later the agent discovers the optimal policy to periodically transmit which can be seen by the checker-colored pattern in the bottom right plot. This example shows that a reinforcement learning agent can learn a simple frequency spectrum usage pattern.

B. Two Learning Agents

The scenario with two learning agents is more difficult than the first. For the first case there is a stationary pattern in the environment that the agent needs to learn. However, in this scenario both agents are learning at the same time. We can run into issues where agents learn both to transmit or not transmit at the same time. Despite these challenges, with the correct hyper parameters we were able to get two learning agents to de-conflict their frequency usage. Furthermore, there is an emergent alternating frequency pattern. We found that the choice of the *transmission normalized* reward type was critical to achieve fairness. The results for this experiment are shown in Figure 4

C. Three Learning Agents, Two Bands

As we increase the number of agents the scenario becomes more complex. In this experiment we test how well we can achieve frequency sharing with three agents and two available frequency bands. The results are shown in Figure 5. We can see that in the beginning agent's learn that they can achieve higher rewards by trying to transmit. As a result two agents receive a lot of collisions as they end up using the same frequency band. One lucky agent picks the band that is free and quickly increases it's value of this state action. Because of the transmission normalized reward and random actions, there is a significant amount of frequency band switching that goes on. In the last plot we can see there is a clear emergent frequency sharing pattern that emerges. We can see noticeable light and dark spots in the agents value of states showing they can anticipate when they should stop transmitting without incurring a switching cost. Two agents learn to perfectly share one frequency band while the third lucky agent has full use of the other band. In this case we have achieved our first goal near optimal frequency utilization. While we are able to achieve perfect sharing by two agents, we still find that a third agent has an unfair advantage. More work should be done to tackle this issue.

It should be noted that this is a single example of this scenario. While I found that this experiment was repeatable in that agents converged to some type of sharing, the steady state was not always the same. Further, the infrequent but present random actions near steady steady still happen and change how the agents act.

D. Actor Critic

In this experiment I implemented an Actor Critic RL model. When given the same single primary user scenario as described in experiment 1, the agent is able to learn the optimal pattern. The results are shown in 6. It can be seen that there is a very smooth transmission to the optimal point. However, I found that I was not able to achieve any amount of fairness in later experiments using Actor Critic. When considering 2 learning agents there was always a single agent that consumed the available spectrum. More experimentation is needed to get the Actor Critic method to work.

VI. Insights Gained

A. Dynamic Spectrum Access Reinforcement Learning

Below I will discuss general insights that were gained from these experiments.

For the choice of reward, we found that only rewarding successful transmissions was not sufficient to encourage learning when agents are trained individually. When agents work towards a shared goal of total successful transmissions it is possible to only reward successes. However, if agents are learning separately, they need to be penalized for collisions, otherwise we run into a prisoner's dilemma scenario where agents are incentivized to transmit all the time. We found that the choice of the *transmission normalized* reward was an effective way of encouraging fairness and learning in a decentralized learning environment.

I experimented slightly with different temporal lengths. I found that 5 was sufficient for two agents but that a larger temporal length of 10 was required when considering 3 or more agents. It makes sense that agents can make better decisions when they consider more temporal information. There is a trade off in increased computation time with a larger temporal length.

In this paper we didn't experiment with the choice of epsilon and epsilon decay. In general these are very important parameters for reinforcement learning and a more in depth hyper parameter search is needed.

B. General Reinforcement Learning

During this experimentation I found that there is an extremely large number of hyper parameters that can be tuned. While I did focus on a subset of them to limit the problem scope, it was still a lot. I found that there are some community tools that can be very useful in RL experimentation. I recommend using *ray tune* [1] for distributed and structured RL hyper parameter search. They provide methods like grid search as well as mechanisms for early stopping of bad trials and evolutionary search for hyper parameters. I became aware of this tool near the end of this study. I did implement a hyper parameter search when can be seen in the repo as an example.

Next, I recommend using an open source tool like *stable-baseline-3* [3] which includes stable implementations of the most popular reinforcement learning models. In my case it made sense to write all my models from scratch for better learning as well as the ability to customize. But, this is a useful tool for getting a baseline.

VII. Conclusion and Next Steps

In this report I have covered an introduction into Deep Reinforcement Learning for Dynamic spectrum access. I implemented several variations of algorithms from other papers and gave an in depth analysis of how the agents learn during the process.

For next steps, I would like to do an more in depth search of the latest RL literature for the current state of the art which may not have been applied for dynamic spectrum access yet. I believe a more thorough search is needed on coordination strategies for prisoner dilemma-like situations. Also, in these scenarios I have only considered single user frequency bands. It is possible for agents to share single bands if there interference is below some threshold. Giving agents the ability to vary their transmission power would add another layer of complexity to the scenario.

Lastly, I would like to test these ideas not only in simulation but with real hardware.

References

[1] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

[2] O. Naparstek and K. Cohen. Deep multi-user reinforcement learning for dynamic spectrum access in multichannel wireless networks. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–7, 2017.

[3] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[4] Y. Xu, J. Yu, and R. M. Buehrer. The application of deep reinforcement learning to distributed spectrum access in dynamic heterogeneous environments with partial observations. *IEEE Transactions on Wireless Communications*, 19:4494–4506, 2020.

Experiment	#1 OnePrimaryUser	#2 TwoAgents	#3 ThreeAgents	#4 OnePrimary User ActorCritic
Number of Agents	2	3	3	2
Number of Frequency Bands	1	2	2	1
Temporal Length	10	10	10	10
Observation Type	aggregate	aggregate	aggregate	aggregate
Reward Type	transmission normalized	transmission normalized	transmission normalized	transmission normalized
Reward History Length	100	100	100	100
RL Model Type	ddqn	ddqn	ddqn	actorcritic
Learning Rate	0.0001	0.0001	0.0001	0.0001
Epsilon Start	0.1	0.1	0.1	0.1
Epsilon Decay	0.00001	0.00001	0.00001	0.00001
Agents Shared Memory	FALSE	FALSE	FALSE	FALSE
Memory Buffer Size	1000	1000	1000	1000
Episode Length	2001	15001	15001	10001
Agent Homogeneity	one periodic	all same	all same	one periodic

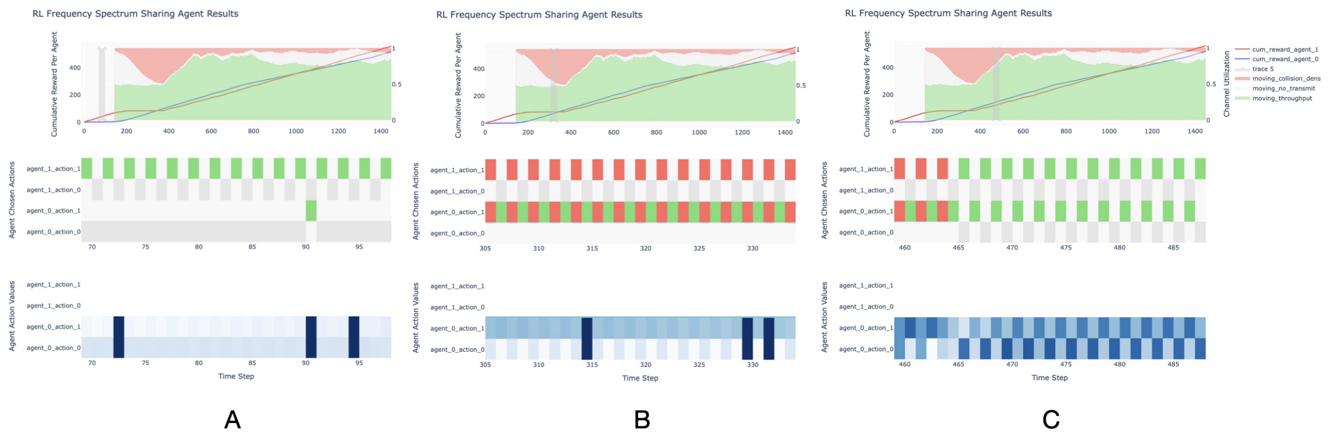

Figure 3. Results from experiment 1. **Top Plots**: Lines show the cumulative rewards per agent over the complete experiment. Shaded backgrounds show a moving average of spectrum status with (green:successful transmissions, red:collisions, gray:unused spectrum). Top plots include a thin shaded gray square representing the zoomed in area depicted in the graphs below. **Middle Plots**: Shows a zoomed in look at what actions agents chose. Colored in squares represent the chosen action (green:successful transmissions, red:collisions, gray:no transmission), **Bottom Plots**: Shows the same zoomed in look but shows the output of each agent's DQN value network. Darker squares represent what the agent sees as more valuable actions at that moment. The darker bars that span across all actions seen in A and B represent instances where a completely random explore action was taken. **A**: A zoomed in look at early time steps from 70 to 100 where the learning agent mostly doesn't transmit. **B**: A zoomed in look at time steps from 305 to 335 where the learning agent always transmits. **C**: A zoomed in look at later time steps from 460 to 490 where the learning agent transmits when the primary user isn't.

[5] C. Zhong, Z. Lu, M. C. Gursoy, and S. Velipasalar. A deep actor-critic reinforcement learning framework for dynamic multichannel access. *CoRR*, abs/1908.08401, 2019.

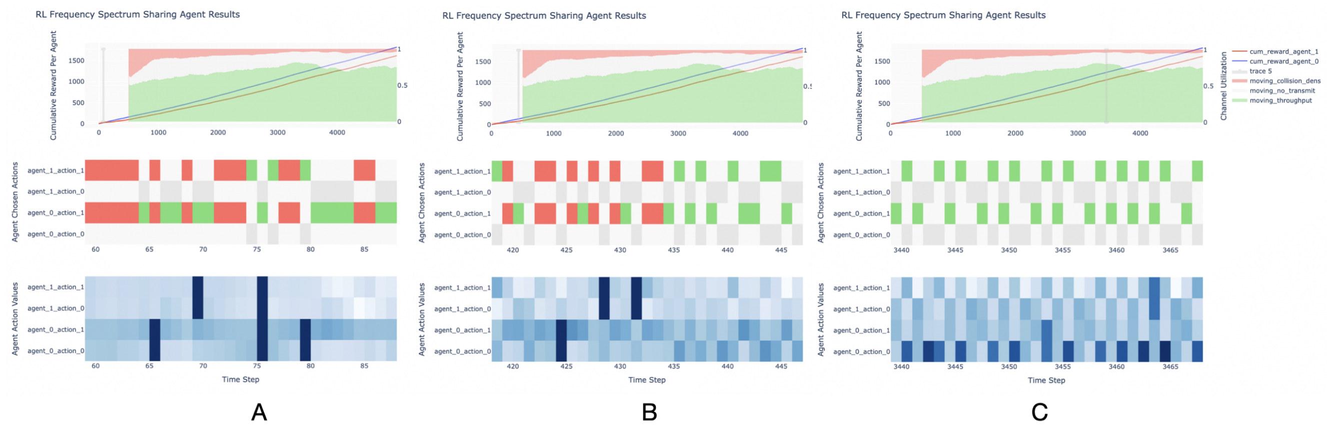

Figure 4. Results from experiment 2. Please read the caption of Figure 3 for a description on how this type of plot is organized.

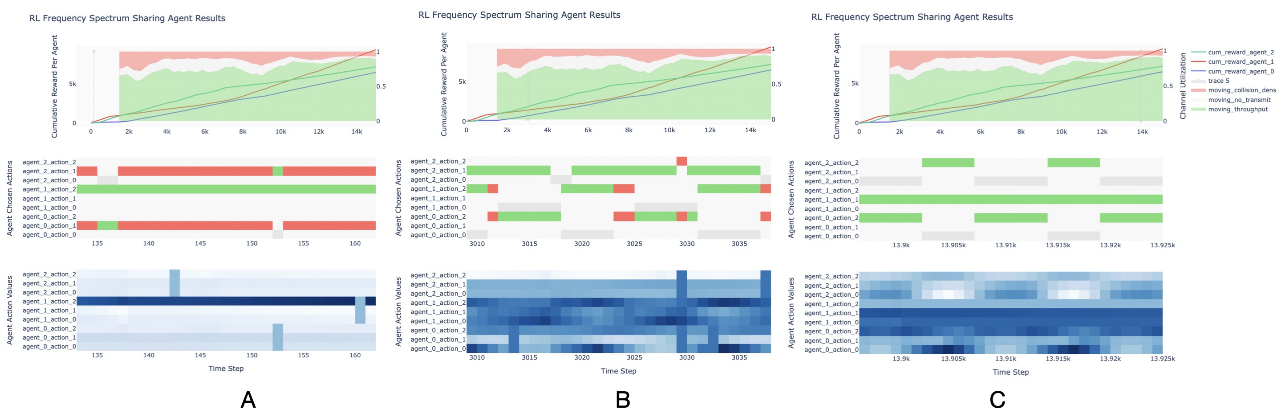

Figure 5. Results from experiment 3. Please read the caption of Figure 3 for a description on how this type of plot is organized.

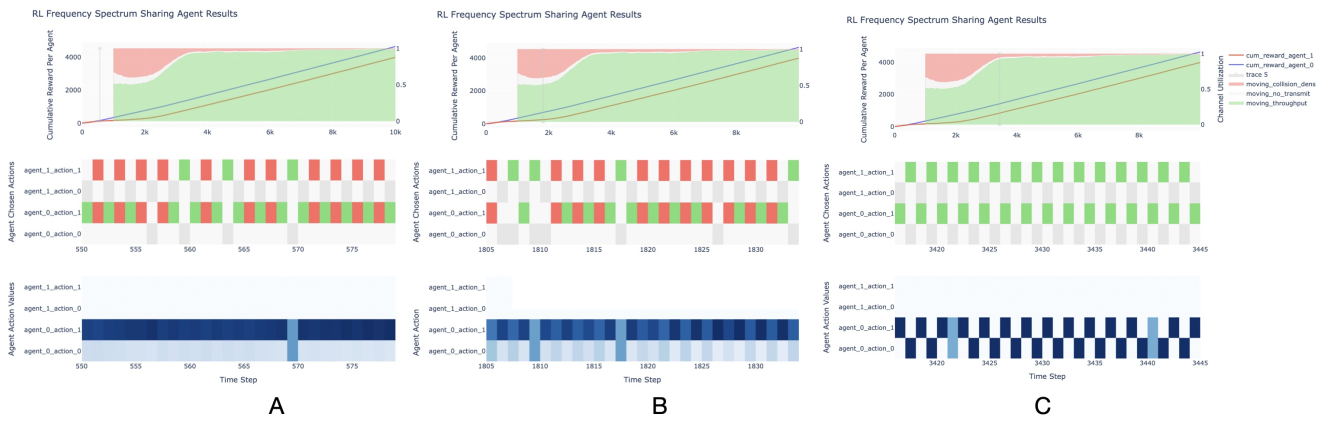

Figure 6. Results from experiment 4. Please read the caption of Figure 3 for a description on how this type of plot is organized.