A Deep Reinforcement Learning Implementation for Dynamic Spectrum Access

Juan Tarrat

***Abstract*—As part of the problem of dynamic spectrum access, reinforcement learning techniques seem a great tool to address the problem of spectrum scarcity in wireless networks. This paper, part of the last submission for an advanced networks course, explores the concepts of spectrum sensing and dynamic spectrum access, as well as a small survey on the use of artificial intelligence in these fields and it's implications. The main work outlined here is a re-doing with small enhancements of an earlier implementation of a deep Q-Learning model for dynamic spectrum access. Experimental results, though not completely satisfactory, demonstrate the promising outcomes that more research on this field could generate.**

I. Introduction

THIS paper corresponds to the final submission of the masters level class *Advanced Computer Networks*, in which we were asked to write a technical report on one of the two projects we have worked on earlier this semester. The project I am discussing is an implementation of deep reinforcement learning for dynamic spectrum access in multi channel wireless networks.

Spectrum sensing in wireless networks refers to the ability of agents in a network to sense the radio spectrum, decide about the state of the channels, and make decisions on their communication parameters based on the state of the network [1]. In 2020 more than 50 billion devices were connected and demanding access to the internet. Spectrum sensing is a cognitive radio technique developed to address the problem of spectrum scarcity. The main objective of radio cognitive techniques like spectrum sensing is to reduce the interference of users in the same network [2].

By using spectrum sensing techniques, we are able to maximize the throughput of a given network in which different actors try to transmit information back and forth. This paper focuses in dynamic spectrum access technique to take an action on a spectrum sensing scenario. Dynamic spectrum access models can fall within 3 categories: dynamic exclusive use model, open sharing model, and hierarchical access model [3]. For this paper, the network being simulated falls withing the hierarchical access model. However, an assumption is made that there is no different hierarchy among different users of the network being simulated.

With recent advancements in artificial intelligence and machine learning, a growing tendency of using these techniques in spectrum sharing and spectrum sensing is being observed. Machine learning techniques in cognitive radio can be divided into three categories: supervised learning(SVM, KNN, Naïve Bayes, etc.), unsupervised learning (K-Means, PCA, GMM, Genetic Algorithm, etc.), and reinforcement learning (Markov Decision Process and Multi-agent learning). The reason behing the use of these techniques in cognitive radio is that these algorithms can solve complex problems by learning, reasoning, and making decisions. All this combined increases the automation level, perception capability, and network flexibility [4].

However, the use of these techniques is not always perfect. In fact, using AI and machine learning poses some challenges. The use of AI in cognitive radio may be hard due to the need of training data, that may be scarce in wireless environments. In addition to this, there are some threats that ought to be mitigated regarding security. As described in [5], there is a variety of ways attackers can compromise the state of a network. One example is dynamic spectrum access attacks, in which the attacker simulates the signal of a primary user thus triggering the algorithm and preventing the secondary user from transmitting any information. In addition to this, as it will be explained later in this paper, there is an added drawback, specially in deep reinforcement learning, that limits the performance of the algorithms. That limitation is the heavy computational power needed to train these sort of networks, which as it is the case in this implementation, if no cloud service is available, performance becomes very limited.

The technique discussed in this paper is reinforcement learning. Reinforcement learning is one of the three paradigms in machine learning (the other two are supervised and unsupervised learning). The main difference between reinforcement learning and the other two paradigms is that the former does not need training data to work [6]. Instead, what happens is that an agent learns to take actions on an environment based on the state of that environment to maximize a reward, finding a balance between exploration and exploitation.

One of the most widely used reinforcement learning algorithms is Q-Learning. In Q-Learning, the agent learns what action to take based on the Q-value each action has at each timestep based on the state of the environment using a lookup table. However, as the problem size increases, it becomes very inefficient. That is when the idea of Deep Q-Learning comes into play. With the use of deep neural networks, the user is able to output an action from the partial observation of the environment.

A. The idea

This paper implements a deep Q-Learning algorithm for dynamic spectrum access following the work done in [7] where the authors used deep Q-Learning to maximize the throughput of a network with multiple users and shared resources. The rules of the game are the following: in a network with C channels and U users, where each user u_i wants to transmit on a certain channel c_i, maximize the reward R where r_i is the reward each user gets if it is the only user to attempt a transmission over channel c_i.

To achieve this, the network must learn to select which channel to transmit in based on previous experiences and on the current state of the environment. As a mild improvement on the experiments done by Cohen and Naparstek in [7], this work shows performance of the network under different settings, like different users and channels, and different architectures of the deep networks.

II. Related Works

Since this is a topic of increasing importance, the amount of work done is also increasing. The reality is that the use of AI for cognitive radio has been around for a while. The work in [8] (2013) developed a cognitive radio model based on an artificial neural network. The model predicted the channel occupancy of a TV band. They were able to show successful training of the network and thus increase the bandwidth efficiency of the specific channel.

More relevant work regarding dynamic spectrum access using deep reinforcement learning technique is shown in the work done in [9]. Here, the authors explored a very thorough scenario that involved a DSA (Dynamic Spectrum Access) network consisting of two channels, four users, and four primary users (PUs). While DSA users can use different channels, PUs have priority. They tested this simulation scenario using four different Q-Learning implementations: Q-Learning, feed forward network DQN with one and two hidden layers, and echo state network (ESN) based DQN. The ESN based DQN is a form of deep reinforcement learning that instead of using a normal deep network, uses an ESN. An ESN is a type of RNN in which only the weights of the output layer are trained. The use of RNN in DQN has the advantage of temporal correlation attribute. The authors were able to show that the ESN based DQN model outperforms the rest of networks in rate, reward, and interference. Naturally, the poorest performance was delivered by the Q-Learning model, which does not use deep learning techniques.

Lastly, it is worth to mention again the work done in [7] and explain it more. The authors in this work also implemented DQN to effectively allocate users over a wireless network. However, their simulation was not as thorough as the one in [9], since all the users have the same priority. They compared their implementation with the ALOHA protocol. The ALOHA protocol states how different users can access different channels in a network without interfering with each other. For their model, instead of using ESN as before, they simply used a LSTM cell to store previous experiences and thus make training more efficient. Another improvement they made over conventional DQN is the use of dueling DQN. This was done following the work in [10] to address the observability problem. In essence, the Q value for each selected action is a function that depends on the average Q-value of the input state and the advantage of taking that action. As opposed to the work in [9], here the authors made use of very powerful computational platforms to train their model (100,000 vs 5,000 episodes). The results were excellent, achieving as high as a 80% channel throughput by the end of training. For reference, the ALOHA protocol achieved a constant 38%. This work is the most relevant for the simulations described in this paper. All the architecture of the network, as well as the training settings are based from the work in [7].

III. System Model

The model simulates a wireless network of U users sharing C orthogonal channels. A real life example of a model like this one is OFDMA, as observed in Figure 1. In a scheme like this one, multiple users can transmit over a wireless network in the same time slot by choosing which channel to transmit. At each time slot, each user selects a channel to transmit and will attempt a transmission on that channel with a probability p. When a packet is transmitted, the user that has sent the package receives an ACK signal. The goal of the network is to maximize the network throughput without user coordination or communication. There are no primary or secondary users involved, meaning that each user will have the same priority for transmitting on a given channel.

The network environment has a set of $\mathcal{U} = \{1, 2, ..., U\}$ users and a set of $\mathcal{C} = \{1, 2, ..., C\}$ orthogonal channels. At each time slot, each user selects a channel on which to transmit its data and will transmit if a certain probability is satisfied. The users are assumed to be backlogged, meaning that they always have a packet to transmit. In order for a transmission to be successful, channel c needs to be the only channel in which a user has attempted a transmission, otherwise no transmission occurs. A user will receive an observation o at time slot t. If the transmission has been successful $o(t) = 1$ otherwise $o(t) = 0$ [7].

Actions are defined as the channel the user decides to transmit on. Thus, mathematically, a user u will take an action $a_u(t) \in \{0, 1..., C\}$ at time slot t. When $a(t) = 0$ the user u has chosen not to transmit on any channel. There is a variety of reasons for this, like managing the congestion level in the network. This can be useful in times where the network has limited resources that could compromise the rest of transmissions if fully utilizing all the spectrum. Similarly, when $a_u(t) = c$ where $1 \leq c \leq C$, the user u has decided to transmit over channel c. The model makes use of a history of variable size $\mathcal{H}_u(t)$ that stores the state of the network at previous time slots [7]. The state of the network is defined as $\mathcal{S}(t) = (\{a_u(t-1)\}, c(t), \{o_u(t)\})$ and is a set of length U stating the partially observable state of the network for each user u. For each user, $a_u(t-1)$ is a one-hot vector of the channel chosen for transmission; $c(t)$ is the binary observation of the channels at time t where $c_i = 0$ if channel is used and $c_i = 1$ if channel is not used; last element is the binary observation $o(t)$ of each user as explained in the previous paragraph. $\mathcal{H}(t)$ is thus a collection of states and is used as input to the deep network.

After each action, users get a reward. Let $r_u(t)$ be the reward the user obtains at a certain time slot t. The reward will depend on the user u action as well as the other users. This is because if two users attempt to transmit over the same channel the reward will be affected. The main objective of the users in the network is to learn what actions to take to maximize accumulated discounted reward $R_u = \sum_{t=1}^{T} \gamma^{t-1} r_n(t)$ where $0 \leq \gamma \leq 1$ and T is the number of episodes the game will last by finding a policy σ_u. This is expressed as

$$\max_{\sigma_u} \mathbb{E}[R_u|\sigma_u] \text{ [7]} \tag{1}$$

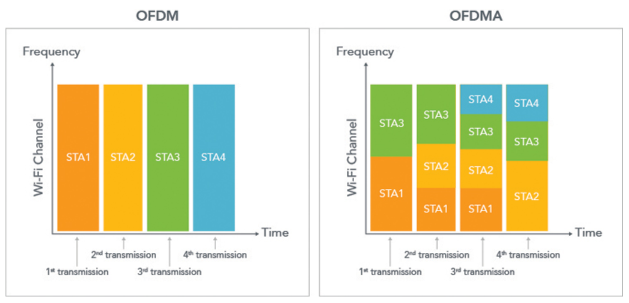

Fig. 1: OFDMA network scheme [11]

IV. Algorithm Development

This section discusses the implementation of the deep reinforcement learning algorithm used to solve the problem stated in section III, and more specifically, to solve (1). We first introduce the term deep Q-Learning, then we explain each of the sections in the network applied to our specific problem, including the training process.

A. Deep Q-Learning

The main idea behind reinforcement learning is to teach an agent to take actions on an environment in order to maximize a reward. The user performs an action based on its current observation of the network and learns a policy $\pi(a_t|s_t)$[12] where a_t is the action at a specific time slot and s_t is the observation of the user at that time slot. With this policy the user receives a reward and an observation for the next time slot.

Q-Learning is the simplest form of this algorithm. In Q-Learning, the network learns the optimal Q-function associated with a certain state. The way it does this is by following the equation

$$Q^*(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a')] \tag{2}$$

where Q^* is the Q-function with an observation s and action a as inputs that returns the expected value of a the sum of the reward r and the expected reward for the next state times a discount factor γ [12]. By following this process for a lot of episodes, convergence is expected and thus optimal actions can be taken for certain observations in the environment. Implementations that use this approach make use of a lookup table for Q-values.

Since in this specific problem it would be very inefficient to use simple Q-Learning, a more sophisticated method is used: deep Q-Learning. In deep Q-Learning a deep neural network is trained so that it learns to approximate the Q-function. The network uses a greedy policy that selects an action with probability ϵ and a random action with probability $\epsilon - 1$. This is done to ensure a wide coverage of the action space and is referred to as exploitation vs exploration. In this implementation, the probability ϵ is calculated as follows:

$$\epsilon = \max z_{stop}, \epsilon * d \tag{3}$$

where z_{stop} is the value at which ϵ is not longer going to be updated and d is the decay rate. Deep Q-Learning makes use of gradient descent to update the weights of the neurons.

B. Architecture of the Implemented Network

This subsection describes the architecture of the model built in this specific implementation. A schematic of the model can be seen in Figure 2.

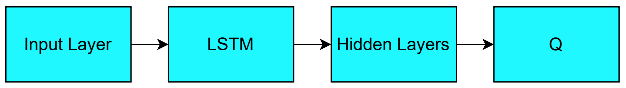

Fig. 2: Architecture of the model implemented.

1) Input Layer: The input $x_n(t)$ is a vector of size $2(C+1)$ which corresponds to the state size for each user after an action has been taken. The first $C+1$ values correspond to the selected channel of the user in in the previous time slot. If the first entry is set to 1, it means that the user has not transmitted at a specific time slot. The next C entries are the capacity of each channel, and the last entry will be set to 1 if the transmission of that specific user has been successful.

2) LSTM Layer: The LSTM maintains an internal state of observations over time. The memory stores experiences; an experience is a tuple of a state, an action vector, a reward vector for that action, and the next observed state after taking the action. The layer is in charge of learning how to aggregate experiences over time. This is a common practice in reinforcement learning implementations to increase efficiency and stability in training.

3) Hidden Layers: At initialization, the network takes a list of user-defined length that contains the neurons of each hidden layer. This offers a lot of flexibility to run different simulations under different configurations in an easier manner.

4) Output Layer: The output layer of the network is a vector of size $C+1$. Each entry corresponds to the estimated Q-value if the user takes an action. Since there are C channels, there is a need of an extra element in the vector to represent when the user does not transmit at a time slot t. The $(k+1)^{th}$ entry is the Q-value for transmitting on channel k.

C. Training the Network

This section shows the function that trains a specific DQN with specified parameters. All the users are trained in the same unit in an offline manner. The training loop takes as inputs the following parameters:

- Episodes: I
- QNetwork: Q
- Environment: E
- Memory: M
- History Input: H

Algorithm 1 Training Loop

```
procedure TRAINDQNETWORK($I$, $Q$, $E$, $M$, $M$, $H$)
    for $i \leftarrow 1, I$ do
        $a \leftarrow E$.sample()
        $s \leftarrow$generateState($a$, $E$.step($a$))        ▷ Generate state
        $\epsilon_i \leftarrow$rand()
        if $\epsilon_i < \epsilon$ then
            $a \leftarrow E$.sample()        ▷ Explore
        else
            $a \leftarrow Q$.predict($H$)        ▷ Exploit
        end if
        $s' \leftarrow$generateState($a$, $E$.step($a$))        ▷ Generate state'
        $r \leftarrow$getReward($E$.step($a$))
        for $j \leftarrow 1, E$.numUsers() do
            if $r(u_j) > 0$ then $r(u_j) = \sum r$        ▷ Cooperative
            end if
        end for
        $M$.add($s$, $a$, $r$, $s'$)        ▷ Store experience in memory
        $b \leftarrow M$.sample()        ▷ Sample batch
        $s' \leftarrow$getNextStates($b$)        ▷ Get next states from $b$
        $q' \leftarrow Q$.predict($s'$)        ▷ Get policy
        $q* \leftarrow r + \gamma * \max q'$        ▷ Get target Q-values
        $Q$.train()        ▷ Train network and update weights
        $\epsilon = \max z_{stop}, \epsilon * d$        ▷ Update $\epsilon$
    end for
end procedure
```

In the code, this function returns a list of rewards that is later used to plot data. In the simulations performed, several networks with different number of users and channels and different architectures were trained over a number of iterations. The training process of a network like this one is very expensive, and for optimal training, the use of a cloud computing service is needed to obtain meaningful results. As it can be observed in the code, the network follows a cooperative policy for rewards: for every time slot t, if the user u has successfully completed a transmission, the reward that user observes is the sum of all the rewards for that time slot:

$$r_u(t) = \sum_{i=1}^{U} r_i \tag{4}$$

V. Simulation

In this section, a discussion of the different simulations ran is exposed. For the simulation setup, the network uses the same hyper parameters (learning rate, batch size, epsilon values, gamma, number of epochs...). There were a total of 6 simulations performed with the following configurations:

- 2 users and 3 channels [64 64]
- 3 users and 2 channels [10]
- 3 users and 2 channels [64 128 64]
- 4 users and 3 channels [10]
- 4 users and 3 channels [64 64]
- 5 users and 4 channels [128 64]

As it has been mentioned previously, training the network, although simple, is a very expensive computational process that requires very powerful machines. A standalone machine would first of all take too long to train, and would run out of memory when the number of episodes exceeds a threshold. The best option is to use a cloud computing service like Google cloud or AWS. Sadly, the simulations ran for this paper only had access to limited free cloud computing services that negatively affected the performance of the network.

The simulations this paper are based on trained the network over 100,000 episodes whereas due to the limitations simulations here had to be ran for 5,000 episodes or after that even the cloud service would run out of memory. Results are reported in the form of network throughput per episode, throughput average for every 500 episodes, and cumulative reward during episodes. Figure 3 shows the throughput of the network at every episode. As it can be observed, for most episodes the throughput is 0; it can be seen though that the networks with the same users and channels but more neurons and hidden layers tend to perform better. Figure 4 shows the average throughput for every 500 episodes. It is clearly observed that no trend can be seen. It can be pointed out though that for the 3 users 2 channels configurations the one with two hidden layers does show a better average throughput overall, proving that a more sophisticated deep network, although more expensive to train, will perform better. The true challenge is to find a balance between performance and time so that one does not completely outbalance the other.

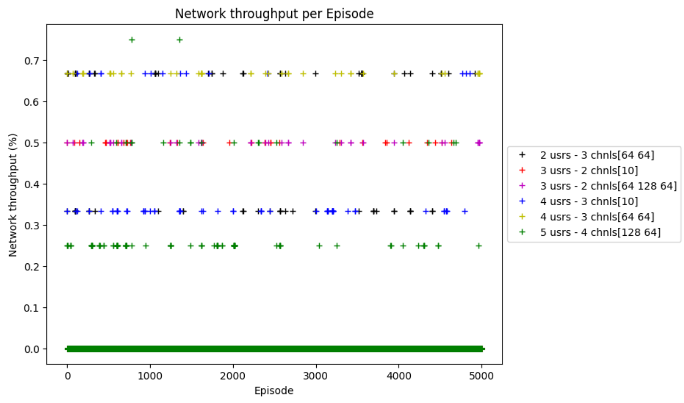

Fig. 3: Network throughput per episode.

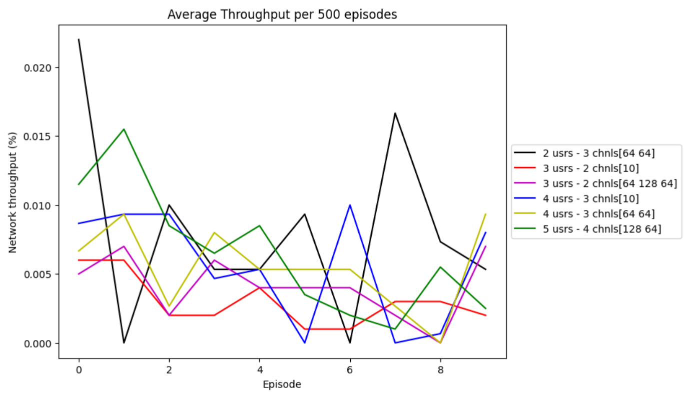

Fig. 4: Average throughput for every 500 episodes

Figure 5 shows the cumulative reward for every episode following the formula $R = \sum_{i=1}^{U} r_i$ where U is the number of users and r_i is the individual reward of each user. Other approaches for policies reward only the users with reward, which is called a competitive policy [7]. Obviously, the more users the larger the reward will always be. There is an anomaly though since the 4 user configuration with one hidden layer has less cumulative reward than the one with two hidden layers. It is the opposite for the 3 users configurations, since the one with two hidden layers performs better.

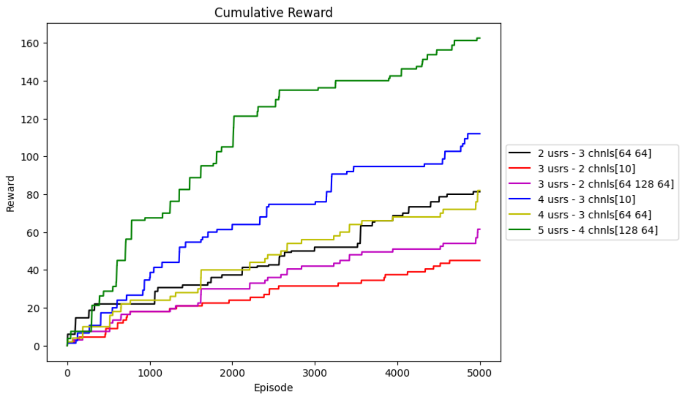

Fig. 5: Caption

Additionally, histograms for the different channel configurations were generated to more easily visualize the spreading of the throughput during the training phases. Figure 6 shows the value spreading for a 2 channel configuration. As it can be observed, most of the values are 0, and only a few are 0.5. The configuration with more hidden layers shows, as expected, better performance.

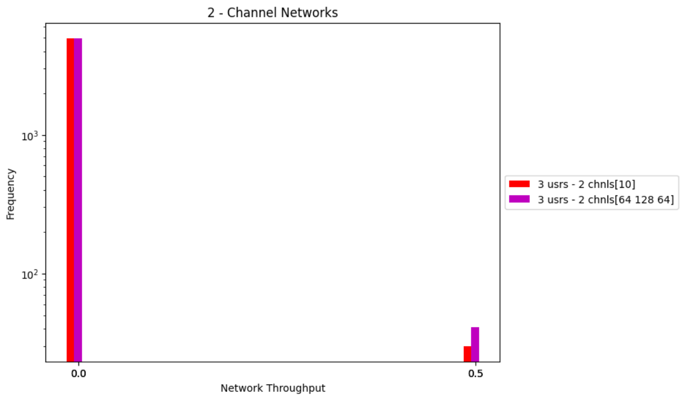

Fig. 6: Throughput for 2 channels configurations

Figure 7 shows the histogram for networks with 3 channels. Again, the network with two hidden layers performed the best for the same number of users. The other configuration uses 2 users so naturally better performance is expected. It is shocking how not even the configuration with more channels than users is still not able to run a whole episode using the full network's capacity.

Figure 7 shows the histogram for the 4 channel network simulation, there is a more even spread of values. However, as in the previous results, the network does not achieve full occupancy.

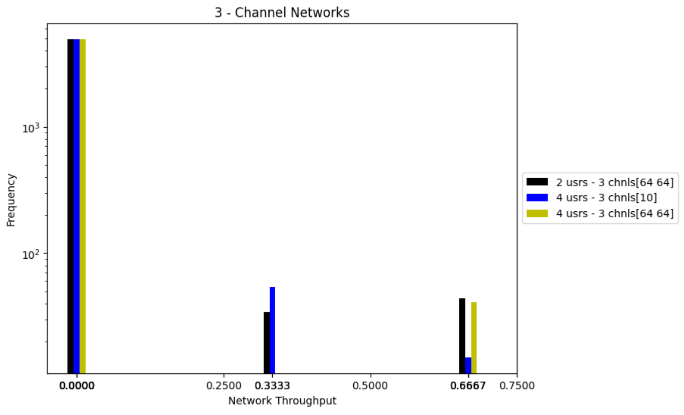

Fig. 7: Throughput for 3 channels configurations

The results are clearly not as good as other simulations in existing works like [7] [4]. This probably mainly due to the limitations during training. Having had access to better and more powerful computing resources, results would most likely been more insightful. There is a chance that the networks spent most of the time just exploring instead of exploiting existing experiences, which would mean that the network would just sample random actions from the environment.

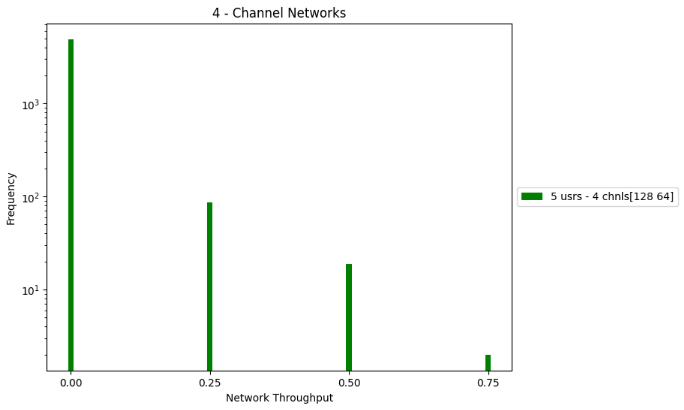

Fig. 8: Throughput for 4 channels configurations

VI. Conclusion

Dynamic spectrum access is an area of expanding interest in wireless networks that has arisen to address the problem of spectrum scarcity. Machine learning techniques like reinforcement learning have the potential of making networks how to use the network resources efficiently. This paper has explored an implementation of the deep Q-learning algorithm to teach agents to transmit over orthogonal channels in a network. Although simulation results are not as good as it would be desired, the implementation shows signs of potential good results if trained using adequate hardware. Future work for this includes implementing another reinforcement learning algorithm like actor-critic, as well as performing simulations in a powerful computing environment.

ACKNOWLEDGMENT

As we reach the end of the semester, and personally the end of my college live, I would like to thank you Dr. Nguyen for such an interesting and insightful course. I am sad I have not been able to extract the maximum out of it and perform as good as I would have wished due to external reasons mainly dealing with the stress of the thesis research work. However, I am still very happy to have chosen this course because I have been introduced to concepts very new to me not only in the field of computer networks, but also in the field of AI and machine learning.

REFERENCES

[1] Y. Arjoune and N. Kaabouch, "A comprehensive survey on spectrum sensing in cognitive radio networks: Recent advances, new challenges, and future research directions," vol. 19, no. 1. Publisher: Multidisciplinary Digital Publishing Institute (MDPI).

[2] Y. Zhang and J. Li, "Study on spectrum allocation and optimization of wireless communication networks based on sfoa," *Wireless Communications and Mobile Computing*, vol. 2021, p. 1–11, 2021.

[3] Q. Zhao and B. M. Sadler, "A survey of dynamic spectrum access," vol. 24, no. 3, pp. 79–89. Conference Name: IEEE Signal Processing Magazine.

[4] D. Janu, K. Singh, and S. Kumar, "Machine learning for cooperative spectrum sensing and sharing: A survey," *Transactions on Emerging Telecommunications Technologies*, vol. 33, Sept. 2021.

[5] T. Clancy and N. Goergen, "Security in cognitive radio networks: Threats and mitigation," pp. 1 – 8, 06 2008.

[6] "Reinforcement learning - Wikipedia — en.wikipedia.org."

[7] O. Naparstek and K. Cohen, "Deep multi-user reinforcement learning for dynamic spectrum access in multichannel wireless networks," pp. 1–7.

[8] S. Pattanayak, P. Venkateswaran, and R. Nandi, "Artificial intelligence based model for channel status prediction: A new spectrum sensing technique for cognitive radio," vol. 2013. Publisher: Scientific Research Publishing.

[9] H. Song, L. Liu, J. Ashdown, and Y. Yi, "A deep reinforcement learning framework for spectrum management in dynamic spectrum access," vol. 8, no. 14, pp. 11208–11218. Conference Name: IEEE Internet of Things Journal.

[10] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2016.

[11] "Wi-Fi 6's OFDMA Challenges Make Verification Crucial — rfglobalnet.com." [Accessed 26-11-2023].

[12] "Introduction to RL and Deep Q Networks—TensorFlow Agents — tensorflow.org." [Accessed 25-11-2023].