American Journal of Engineering and Applied Sciences

Original Research Paper

Combining Q-Learning and Multi-Layer Perceptron Models on Wireless Channel Quality Prediction

[1,2]Andrea L. Piroddi and [1]Maurizio Torregiani

[1]*Department of Computer Science, University of the People, Pasadena, California, USA*
[2]*Department of Computer Science, Università di Bologna, Campus di Cesena, Italy*

Article history
Received: 17-01-2021
Revised: 01-03-2021
Accepted: 09-03-2021

Corresponding Author:
Andrea L. Piroddi
Department of Computer Science, University of the People, Pasadena, California, USA
Email: andrea.piroddi@uopeople.edu

Abstract: One of the most complex challenges that wireless communication systems will face in the coming years is the management of the radio resource. In the next years, the growth of mobile devices, forecast (CISCO, 2020), will lead to the coexistence of about 8.8 billion mobile devices with a growing trend for the following years. This scenario makes the reuse of the radio resource particularly critical, which for its part will not undergo significant changes in terms of bandwidth availability. One of the biggest problems to be faced will be to identify solutions that optimize its use. This work shows how a combined approach of a Reinforcement Learning model and a Supervised Learning model (Multi-Layer Perceptron) can provide good performance in the prediction of the channel behavior and on the overall performance of the transmission chain, even for Cognitive Radio with limited computational power, such as NB-IoT, LoRaWan, Sigfox.

Keywords: Q-Learning, Network Research, OpenaAI Gym, Network Simulator, ns-3, Supervised Learning, Multi-Layer Perceptron

Introduction

The current communication networks are rather complex dynamic systems; on the other hand, the simulation tools we have, to estimate the behavior of these architectures are based on simplified models that are often unable to reproduce the interaction of the multiple components involved such as the presence of interferers and phenomena such fading, moving obstacles, atmospheric events and last but not least the characteristics of the surrounding environment that can have a negative impact on the parameters of our system, such as frequency, amplitude, delay, etc. It is also true that networks today are able to produce a huge amount of measurement data and metadata, which if properly exploited could improve the management and interaction between the different elements in the network (Samek *et al.*, 2017). Machine learning algorithms, Reinforcement Learning specifically, are particularly well suited for this purpose. The idea is to change the paradigm used so far, in which the goal is to adapt the transmission to the change in the characteristics of the channel in a new methodology that aims to predict what the characteristics of the channel will be in the instant preceding the transmissive event. In the following sections we introduce the concept of Cognitive Radio, then, we will show a Supervised Learning model, applied to an indoor context in which the system is able to predict the behavior of the channel inside the premises and to adapt some transmission parameters to guarantee a constant BER value. Finally, referring to the precious work done by (Gawłowicz and Zubow, 2019) in which it is proposed to combine the two simulation tools Network Simulator (NS-3) and OpenAI-Gym we present an optimized Q-Learning algorithm, which allows the agent to predict the behavior of the environment when sudden interference occurs in the system and consequently implementing the correct policy, in an Unsupervised Learning set. Having a better link quality means having higher ratio of successful reception and therefore a more reliable communication. The original contribution of this paper is the following: By appropriately combining two Machine Learning methodologies, it is possible to predict the behavior of the radio channel with a low computational cost, making this approach suitable for application in environments where terminals have limited computing capacity, as in IoT systems, LoRaWan and Sigfox. This translates into longer battery life and the possibility of increasing the number of terminals in the area served by a single node.

Background

"A Cognitive Radio is the application of intelligent processing and adaptation to a wireless communications system" (Rondeau and Bostian, 2009). The basic idea is to make our network element an entity capable of learning, through the observation of environmental parameters, the behavior of the transmission channel and predicting its variations by acting in such a way as to optimize its performance in terms of throughput, power, coding scheme, energy consumption and at the same time minimizing interference to other devices.

In Fig. 1 we show an example of the policy that the agent implements, foreseeing a variation of the characteristics of the transmission channel. Since long time, in wireless communications we have learned how to describe the channel used for transmission, using different parameters, such as the operating frequency, the type of transmission medium (e.g., air, water), the type of environment (e.g., indoor, outdoor, urban, etc. …), the relative position of communicating parts (e.g., line of sight, not line of sight). The physical layer technology implemented in the transmitter and receiver includes blocks, such as the antenna, the frequency shifter, the sampler, the synchronizer, etc. The link layer is responsible for the correct delivery of the data frame, therefore includes header assembly and disassembly techniques and payload encoding and decoding, as well as mechanisms for correcting and checking errors and retransmissions. While the quality of a link is eventually influenced by a relatively limited number of observations, the so-called set of metrics.

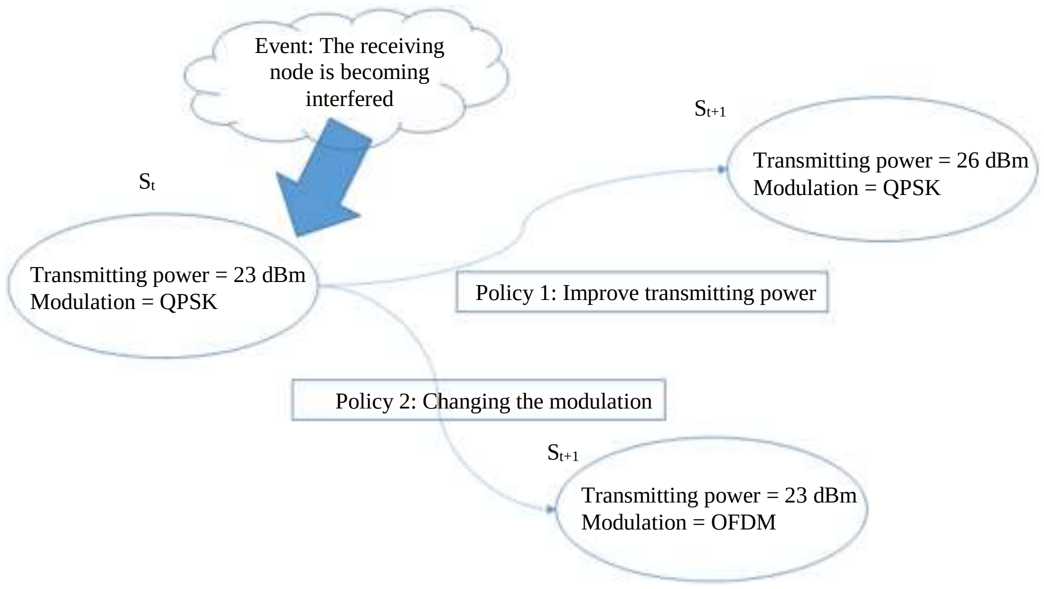

Event: The receiving node is becoming interfered
S_t
Transmitting power = 23 dBm
Modulation = QPSK
S_{t+1}
Transmitting power = 26 dBm
Modulation = QPSK
Policy 1: Improve transmitting power
Policy 2: Changing the modulation
S_{t+1}
Transmitting power = 23 dBm
Modulation = OFDM

Fig. 1: Agent policy

Table 1: Metrics that can be used to measure the link quality

Link quality metrics	Hardware base	Software-base PRR-based	RNP-based	Score-based	Image based	Topological	Sides involved Rx	Tx	Gathering method Passive	Active	Related base-metric(s)
RSSI	✓						✓		✓		RSS, SNR
LQI	✓						✓		✓		Vendor-specific
SNR	✓						✓		✓		RSS, noise floor
BER	✓						✓		✓		-
PRR		✓					✓		✓		PER
WMEWMA		✓					✓		✓		PER, PRR
4B			✓				✓	✓	✓	✓	LQI, PRR, ACK, broadcast
LQ, NLQ			✓				✓	✓		✓	-
ETX			✓				✓	✓		✓	LQ, NLQ
4C				✓			✓		✓		LQI, PRR, SNR, RSSI
TRIANGLE				✓			✓		✓		SNR, LQI
Image-based					✓						
Topological						✓					

Table 1 (Cerar *et al*., 2018) collects the metrics that can be used to measure the radio link quality. Each metric can also be used as an input for another metric. So-called hardware-based metrics, such as Received Signal Strength Indicator (RSSI), Link Quality Indicator (LQI), Signal to Noise Ratio (SNR) and Bit Error Rate (BER) are produced directly from devices and depend on the underlying metrics, such as the Noise Figure, specific to one or another supplier. It is clear, looking at the table, that the number of the independent variables is bounded. However, recently, like additional input, the Topological (surrounding space) feature was taken into consideration, which presupposes the exchange of information on several levels, where the Learning Quality Estimator (LQE) is informed about the distance from the base station (or access point), etc... In this study we considered the development of a model based on topological data and classical metrics.

The classic approaches of channel resources management are based on measurement data report sent by the terminal to the central entity and decision action provided by the central unit to the terminal. These algorithms are managed centrally by the control unit that sends the actions to be performed by the terminal, such as a handover on a different node, or an increase in transmission power, or a change in modulation. This methodology presents a criticality: The device must always be connected with the central unit, otherwise the connection will be disrupted. Any unexpected variation of the radio parameters, such as sudden interference, can cause packet loss and the need to retransmit several times both the payload and the channel control packets. Essentially, the terminal is never autonomous in deciding which action to take in order to maintain the connection. In the event of sudden changes in the surrounding conditions, our terminal must be able to autonomously interpret the data collected and implement a decision that allows it to prevent the loss of the connection with the central unit. This is the reason why we propose the combined use of supervised and unsupervised learning methods in the management of radio resources.

System Architecture

Dataset and Layout

To verify our idea, we used an excellent dataset made available to the scientific community by Gonzalez-Ruiz - University of New Mexico (Gonzalez-Ruiz *et al*., 2011). The wireless channel measurements were collected indoor over a floor of the ECE building. at UNM along several routes. Figure 2 shows the floor plan as well as the regions where the measurements were taken. The triangular symbol shows the position of the transmitter. The position of the origin is also marked. Measurements are made in different regions marked by R and were collected with a router acting as the Transmitter (Tx) and a Pioneer robot that carries a WiFi card acting as a Receiver (Rx). Both transmitter and receiver are omni-directional. The WiFi card is an Atheros ar5006x WiFi card, operating at 2.4 GHz. The coordinates (x, y, z) of the origin are set to be (0, 0, 0). The unit used in this document and the data files is meter. The transmitter's location is (0.115, 0.11, 1.5). There is a total of 16 regions of measurements (R1-R16) and each region contains several routes of measurements. There is a total of 67 routes. The total number of measures is 12463, that is a solid dataset to work with. The data are in the format shown in Table 2.

Column 4 is the measured RSSI (Received Signal Strenght Indicator) of the signal in dBm.

Having a clear picture of our environment, we supposed the occurrence of an interferential source in a random point of our layout. The interference source will be a stationary signal over time, with transmission power of 0 dBm, center frequency and bandwidth the same as those used by the router and fixed position. We thought the interfering signal subject to the fading and path loss according to the Multi-Wall indoor Model (Publications Office of the EU, 1999) that is:

$$L_{dB} = 20\log\left(\frac{4\pi R}{\lambda}\right) + L_c + \sum\nolimits_{i=1}^{N_{type}} N_{wi}L_{wi} + N_f L_f \tag{1}$$

Where:

L_c	=	Constant loss
N_{wi}	=	N. of penetrated walls of type *i*
L_{wi}	=	Loss of walls of type *i*
Nf	=	N. of penetrated floors
N_{type}	=	N. of wall types
L_f	=	floor loss

Since the loss due to floor penetration experimentally appears to be non-linear with the number of crossed floors, then an alternative version of the MWiM model has been proposed:

$$L_d B = 20\log\left(\frac{4\pi R}{\lambda}\right) + L_c + \sum\nolimits_{i=1}^{N_t ype} N_{wi}L_{wi} + N_f^{\left(\frac{N_f+2}{N_f+1}-b\right)} L_f \tag{2}$$

Typical parameter values are: L_c = 0 dB, L_{wi} = 3-5 dB, L_f = 15-20 dB, b = 0.46.

Table 2: Format of dataset

Column 1	Column 2	Column 3	Column 4
x position of the receiver (m)	y position of the receiver (m)	Distance between transmitter and receiver (m)	Received signal strength (dBm)

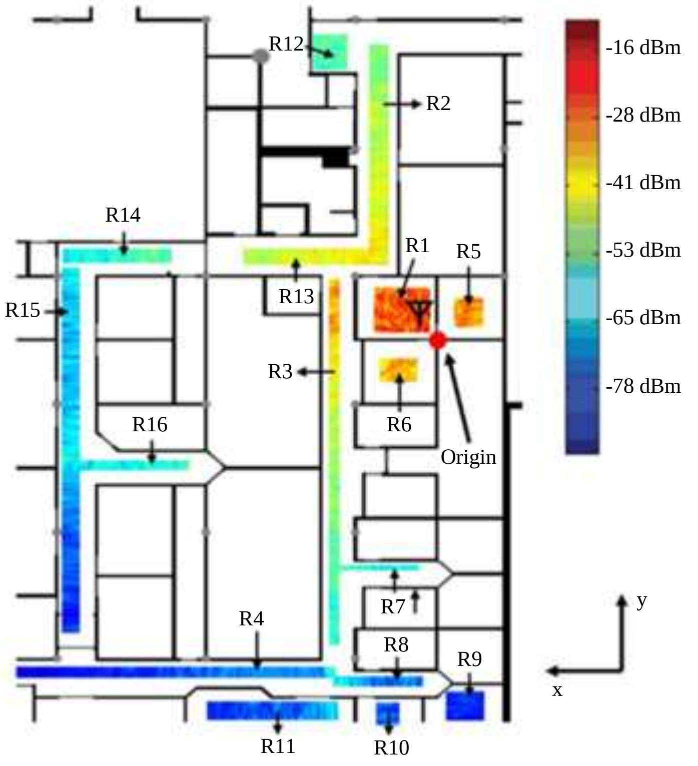


Fig. 2: ECE building basement. At University of New Mexico - a colormap of the measured RSSI is superimposed on the blueprint

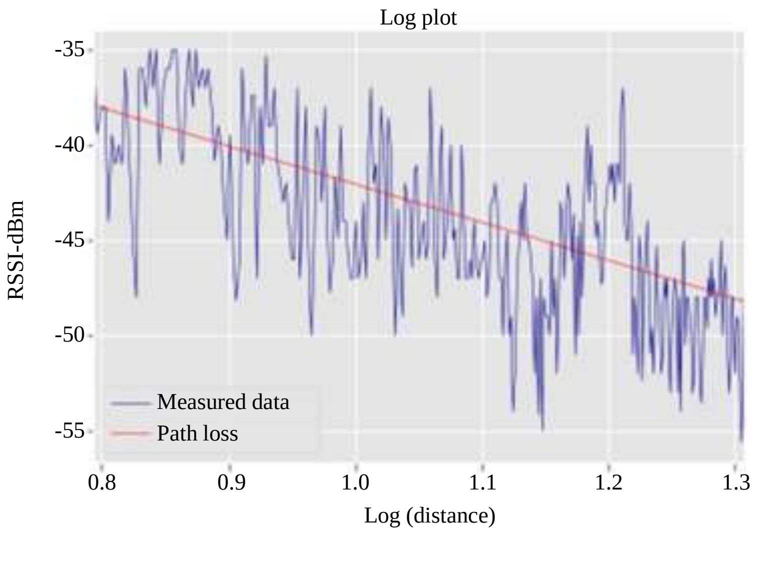


Fig. 3: RSSI-measured data and theoretical path loss

In Fig. 3 we can see the trend of the RSSI measured in one of the paths in Region 2.

SINR Modeling

SINR is usually defined for a specific receiver (or user). For a receiver positioned at some point x in space, its corresponding SINR value is given by:

$$SINR(x)\frac{P}{I+N} \tag{3}$$

where, P is the received signal (of interest) power, I is the power of the other (interfering) signals in the network and N is the noise term, which may be random or a constant. In the following we are going to consider an Additive White Gaussian Noise (AWGN).

The propagation model leads to a model for the SINR (Andrews *et al*., 2010): Consider a collection of *n* transmitters located at points x_1 to x_n in the plane or 3D space. Then for a user located, for example, at $x = 0$, the SINR for a signal coming from *i-th* base station (x_i) is given by:

$$SINR_o = \frac{h_{oo}\rho_o r^{-\alpha}}{N_o + \sum_{i\in\phi}\rho_i h_{io}\left|X_i\right|^{-\alpha}} \tag{4}$$

where, h_{io} is the power fading coefficient of the channel to the receiver of interest "*o*" from node "*i*", ρ_i is the power of transmitter "*i*" and φ is the set of interfering nodes (φ is a subset of all possible transmitters). The desired transmitter is at distance *r* from the desired receiver, while the *i-th* interferer is at distance X_i away. In our case we can consider numerator as the measured RSSI. The component can be seen as the Interference $\sum_{i\in\phi}\rho_i h_{io}\left|X_i\right|^{-\alpha}$ signal that reach our receiver from the interference source (α is the path loss exponent >2) and the term N_o is the Noise Power in the origin, given by:

$$N = 10\log_{10}\left(BN_0\right) + 30 \tag{5}$$

N_0 is the Noise Power Spectral Density given by: $N_0 = k_BT$ (k_B is the Boltzmann's constant: $1.38\cdot10^{-23} J/K$) and *T* is the system temperature (*K*). This means that we can calculate the SINR in each point of the floor. Figure 4 shows the SINR measured in Region 2.

Our goal is to predict the behavior of the transmission channel to choose the policy for optimizing the performance of our system. For simplicity, we will consider the optimization of the throughput. So, the basic idea is to use the most appropriate Modulation and Coding Scheme according to the prediction. To do this it is needed a prediction of the BER. One technique used to determine the quality of a digital transmission system is measuring its Bit Error Ratio (BER). The BER estimate is obtained by comparing the transmitted sequence of bits to the one received and counting the number of errors. The ratio between the bits received in error and the number of total bits received is the BER:

$$BER = \frac{N_{Err}}{N_{bits}} \tag{6}$$

This is a statistical process, so the measured BER only approaches the actual BER if the number of bits tested approaches infinity. In most cases we need only testing if the BER is less than a pre-defined threshold. The number of bits needed will depend only on the BER threshold and on the required confidence level.

Figure 5 (Nordin, 2012) shows how the BER varies as a function of the Dynamic Subcarrier Allocation - SINR based on the type of Modulation and Coding Schemes (MCSs) being used.

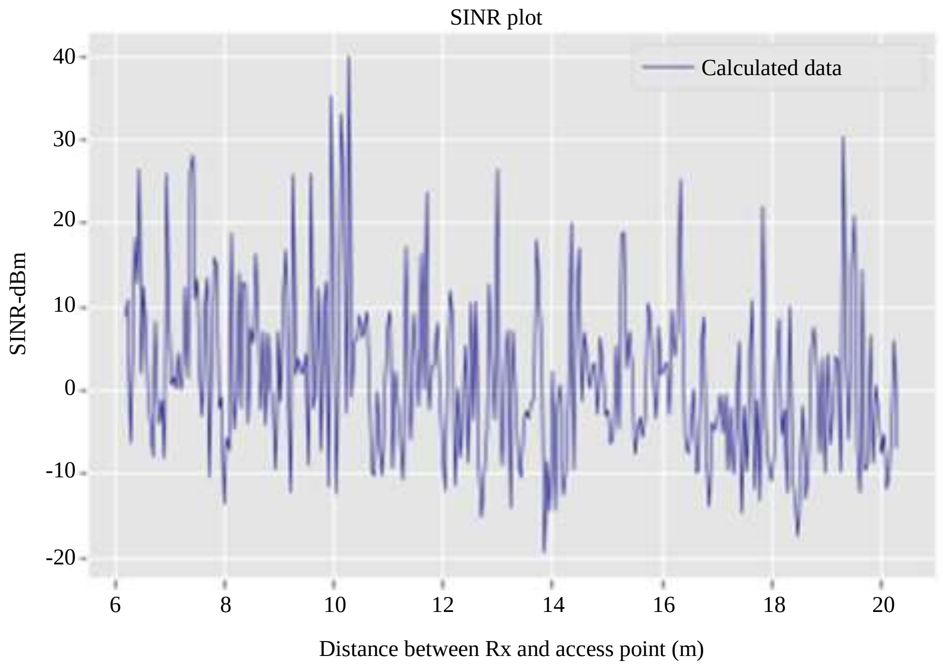


Fig. 4: SINR in region 2

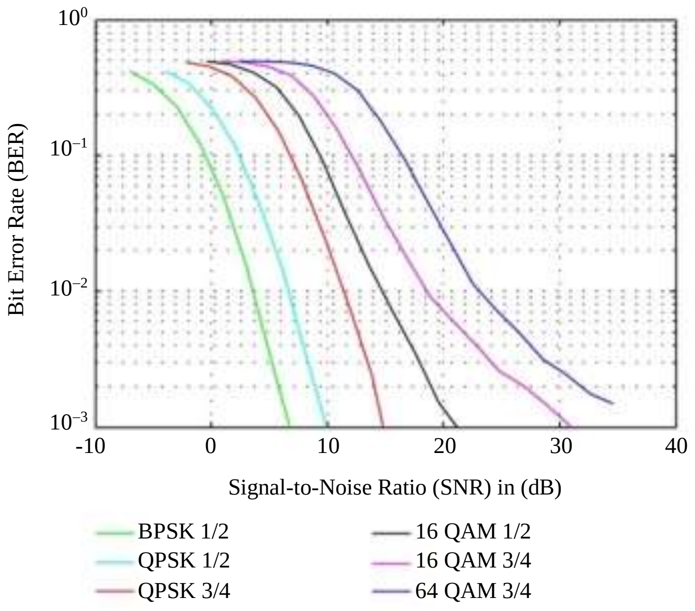


Fig. 5: BER performance of Dynamic Subcarrier Allocation (DSA)-SINR across different MCSs

Multi-Layer Perceptron Model

The Multilayer Perceptron (MLP) is an artificial neural network model (Fig. 6) that maps set of input data into a set of appropriate output data. It is made up of multiple layers of nodes in a direct graph, with each layer completely connected to the next. Except for incoming nodes, each node is a neuron with a non-linear activation function. Multilayer Perceptron uses a supervised learning technique called backpropagation for network training. MLP is a modified version of the classical Linear Perceptron and can differentiate data that are not linearly separable. The fact that it is a supervised neural network clearly suggests that this part of our optimization involves the interaction with a central entity that will update the policy. We will use sigmoid, also known as the logistic function, as the activation function:

$$\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}} \tag{7}$$

the output obtained after the forward extension is known as the expected value ($\hat{y}$).

Learning Algorithm

The learning algorithm is composed by two parts: Backpropagation and optimization. In the backpropagation process a loss function is used to know an estimate of how far we are from our desired solution. Generally, the Mean Square Error (MSE) is chosen as the loss function for regression problems and the cross entropy for classification problems. Given a regression problem its loss function is the mean square error, which squares the difference between the actual (y_i) and the predicted ($\hat{y}_i$) value:

$$MSE_i = \left(y_i - \hat{y}_i\right)^2 \tag{8}$$

The loss function is computed for the entire training dataset and its average is called the cost function C:

$$C = MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 \tag{9}$$

To find the best weights for our Perceptron, we need to realize how the cost function changes in relation to weights and biases. This is done with the help of gradients. So, we need to identify the gradient of the cost function with respect to weights and bias.

We compute the gradient of the cost function C using the partial derivation, with respect to the weight w_i. Since the cost function does not depend directly on the related weight w_i, we use the chain rule:

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial \hat{y}} \times \frac{\partial y}{\partial z} \times \frac{\partial z}{\partial w_i} \tag{10}$$

Equation 11 shows the gradient of the cost function (C) with respect to the predicted value ($\hat{y}$):

$$\frac{\partial C}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 = 2\times\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right) \tag{11}$$

Be $y = [y_1, y_2, \dots y_n]$ e $\hat{y} = [\hat{y}_1, \hat{y}_2, \dots \hat{y}_n]$ the line vectors of actual and predicted values. So, the above equation is simplified as:

$$\frac{\partial C}{\partial \hat{y}} = \frac{2}{n} \times sum\left(y - \hat{y}\right) \tag{12}$$

Equation 13 compute the gradient of the predicted value with respect to *z*:

$$\begin{aligned}\frac{\partial y}{\partial z} &= \frac{\partial}{\partial z}\sigma(z) = \frac{\partial}{\partial z}\left(\frac{1}{1+e^{-z}}\right) = \frac{e^{-z}}{\left(1+e^{-z}\right)^2} = \left(\frac{1}{1+e^{-z}}\right)\\ &\times \frac{e^{-z}}{\left(1+e^{-z}\right)} = \left(\frac{1}{1+e^{-z}}\right)\times\left(1-\frac{1}{1+e^{-z}}\right) = \sigma(z)\times\left(1-\sigma(z)\right)\end{aligned} \tag{13}$$

Equation 14 shows the gradient of *z* with respect to the weight w_i is:

$$\frac{\partial z}{\partial w_i} = \frac{\partial}{\partial w_i}(z) = \frac{\partial}{\partial w_i}\sum_{i=1}^{n}\left(x_i w_i + b\right) = x_i \tag{14}$$

So, we get:

$$\frac{\partial C}{\partial w_i} = \frac{2}{n} \times sum\left(y-\hat{y}\right)\times\sigma(z)\times\left(1-\sigma(z)\times x_i\right) \tag{15}$$

It is theoretically considered that the bias has an input of constant value 1.

Let's now turn to the optimization. Optimization is the selection of the best weights and the perceptron bias. For example, choosing gradient descent as the optimization algorithm, it changes the weights and bias, proportionally to the negative of the gradient of the cost function with respect to the corresponding weight or bias. The learning rate (α) is a hyperparameter that is used to control how much the weights and bias are changed.

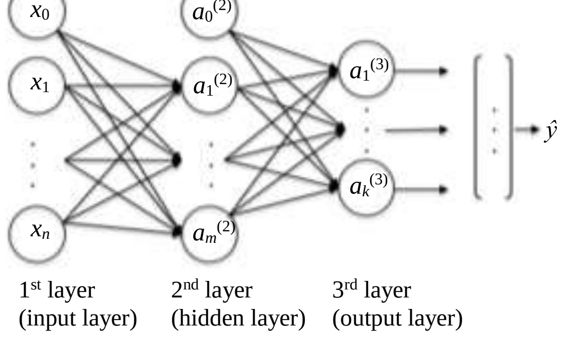


Fig. 6: Multi-layer perceptron model

Weights and bias are updated as follows and back-propagation and gradient descent are repeated until convergence:

$$w_i = w_i - \left(\alpha \times \frac{\partial C}{\partial w_i}\right) \tag{16}$$

$$b = b - \left(\alpha \times \frac{\partial C}{\partial b}\right) \tag{17}$$

Application of the MLP to the Prediction of the MCS

Starting from our indoor environment dataset, we can train the MLP to identify the correct policy for choosing the MCS. As input values we have the position of the receiver, its distance from the access point, the received RSSI levels and the measured SINR levels, as output value we want our system to indicate which MCS to use (Fig. 5) or if it is the case to change carrier. We then build our MLP using Python code and the Scikit-learn library. Scikit-learn is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium-scale supervised and unsupervised problems (Pedregosa *et al*., 2011). To import the dataset and make it available as input to the Scikit-learn MLP we used Pandas. Pandas is an open-source library which provides high-performance and data analysis tools for Python (Pandas Devel. Team, 2020). LP-Classifier trains iteratively, as, at each time step, the partial derivatives of the loss function with respect to the model features are calculated to update the parameters. To prevent the overfitting phenomena, a regularization term can be added to the loss function. The Python code is used to upload data, that are represented as dense numpy arrays of floating-point values and to run the MLP classifier. We run the simulation with different values of both α, the number of hidden layers, the number of nodes in the hidden layers and the number of iterations Fig. 7 shows the MLP Classifier configuration Row).

The chosen classification is the one shown in Fig. 8 considering BER $\leq 10^{-2}$:

Furthermore, we have opted for different configurations both in terms of solver and type. Figure 9 shows some training loss curves obtained with different learning strategies, such as Stochastic Gradient Descend (SGD), Momentum, Nesterov Accelerated Gradient and Adam.

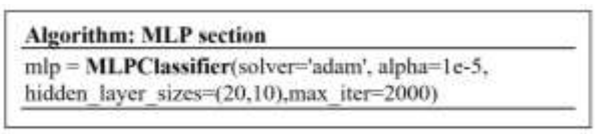
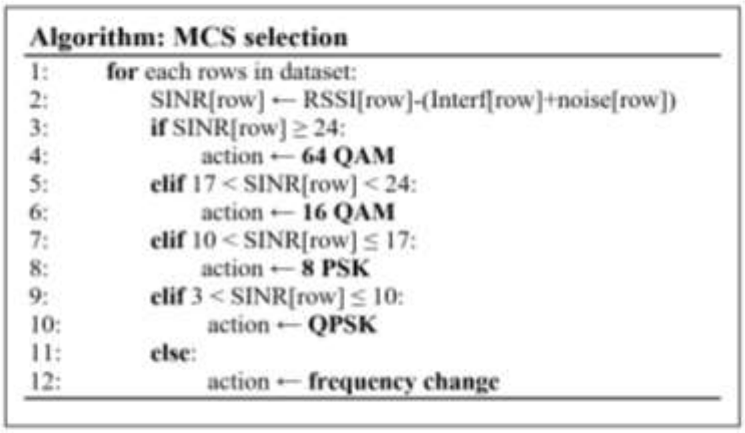

Algorithm: MLP section
```
mlp = MLPClassifier(solver='adam', alpha=1e-5,
hidden_layer_sizes=(20,10),max_iter=2000)
```

Fig. 7: MLP Classifier configuration Row



```
Algorithm: MCS selection
1:   for each rows in dataset:
2:       SINR[row] ← RSSI[row]-(Interf[row]+noise[row])
3:       if SINR[row] ≥ 24:
4:           action ← 64 QAM
5:       elif 17 < SINR[row] < 24:
6:           action ← 16 QAM
7:       elif 10 < SINR[row] ≤ 17:
8:           action ← 8 PSK
9:       elif 3 < SINR[row] ≤ 10:
10:          action ← QPSK
11:      else:
12:          action ← frequency change
```

Fig. 8: Algorithm decision scheme for MCS policy

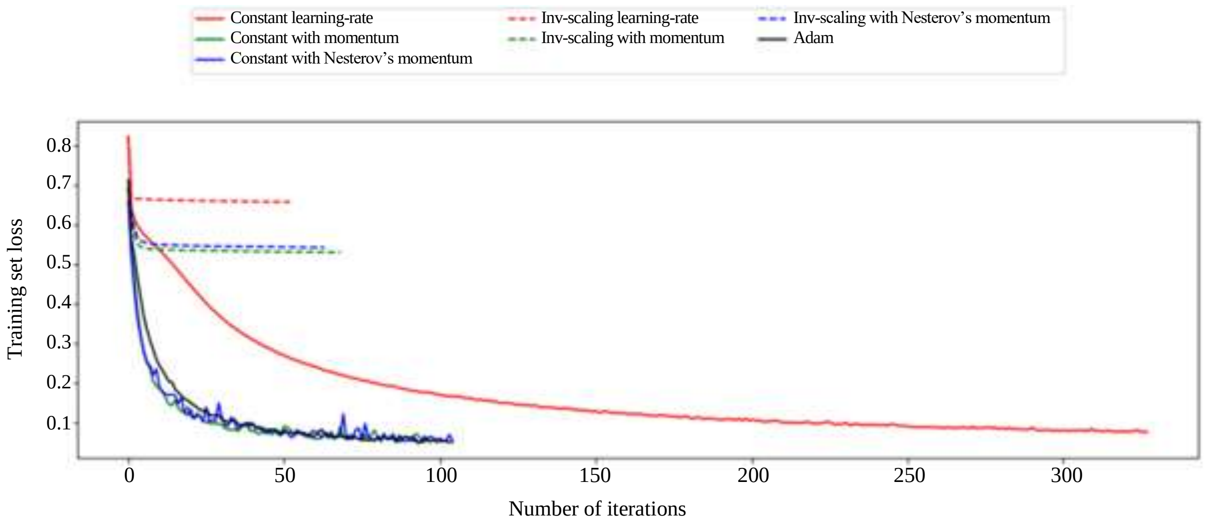


Fig. 9: Training loss curve for MCS choice

Stochastic gradient Descend performs a parameter update for each training sample x^i and label y^i:

$$w = w - \alpha \nabla_w C\left(w; x^i; y^i\right) \tag{18}$$

SGD runs frequent updates with a high variance producing a heavy fluctuation of the objective function.

Momentum is a method that aims to accelerate SGD in the relevant direction by adding a fraction γ of the update vector of the past time step to the current update vector:

$$v_t = \gamma v_{t-1} + \nabla_w C(w) \tag{19}$$

$$w = w - v_t \tag{20}$$

The momentum term increases updates for dimensions whose gradients head in the same directions and decreases them for dimensions whose gradients change directions. The result is that it gains faster convergence and reduced oscillation.

Nesterov Accelerated Gradient (NAG) is a way to provide our momentum an approximation of the subsequent position of the parameters, a rough sign where our parameters are going to be:

$$v_t = \gamma v_{t-1} + \eta \nabla_w C\left(w - \gamma v_{t-1}\right) \tag{21}$$

$$w = w - v_t \tag{22}$$

Adaptive Moment Estimation (Adam) is another method that estimates adaptive learning rates for each parameter. Besides storing an exponentially decreasing average of past squared gradients v_t, Adam strategy keeps also an exponentially decreasing average of past gradients m_t, similar to momentum. For the sake of brevity, g_t is used to denote the gradient at time step t, so

$g_{t,i}$ is then the partial derivative of the objective function to respect the parameter w_i at time step t:

$$g_{t,i} = \nabla_w C\left(w_{t,i}\right) \tag{23}$$

The decreasing averages of past gradient m_t and past squared gradient v_t are computed as follows:

$$m_t = \beta_1 m_{t-1} + \left(1-\beta_1\right) g_t \tag{24}$$

$$v_t = \beta_2 v_{t-1} + \left(1-\beta_2\right) g_t^2 \tag{25}$$

m_t is an estimate of the first moment (the mean) and v_t of the second moment (the uncentered variance) of the gradients, hence the name of the method. To counteract these biases the strategy computes bias-corrected first and second moment estimates:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{26}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{27}$$

Adam works fine in practice and competes to other adaptive learning-method algorithms.

Results of MLP Prediction

Figure 9 shows the training loss curve for the choice of the MCS. We have considered seven different learning strategies:

- Constant learning-rate
- Constant with momentum
- Constant with Nesterov's momentum
- Inv-scaling learning-rate
- Inv-scaling with momentum
- Inv-scaling with Nesterov's momentum
- Adam

The convergence is reached after fifty iterations with Adam strategy that appears as the most appropriate for this scenario.

Table 3 shows the classification report using ADAM learning strategy.

This shows that the level of accuracy is high, although we must consider that this environment is far from being realistic. We should take into account other interfering elements and moving obstacles inside the set, in order to make the scenario more accurate. On the other hand, it is true that the more interferers there are, the greater the contribution of measures that will be made available to the central entity to recalculate the policy, because each interferer works also as a data source. In any case, the result obtained provides some interesting food for thought: The accuracy of such a system can be different changing the learning strategies. For example, using a constant learning-rate policy, the obtained score is 0.984113, while using an inv-scaling learning-rate the score is 0.743400 and with inv-scaling with Nesterov's momentum the score is 0.770200. Furthermore, from Table 3 it is noted that the most critical cases, i.e., those in which it is necessary to be reasonably sure of the prediction, are the two cases with less uncertainty. Figure 10 shows the MLP's policy distribution for the 64QAM Modulation and Coding Scheme.

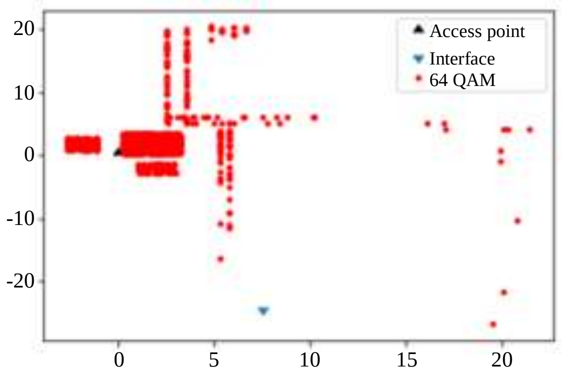


Fig. 10: 64QAM distribution

Table 3: Classification report using ADAM learning strategy

	Precision	Recall	F1-score	Support
#64QAM	1.00	1.00	1.00	499
#16QAM	0.99	0.98	0.98	310
#8BPSK	0.96	0.99	0.97	269
#QPSK	0.98	0.96	0.97	188
#change carrier	1.00	1.00	1.00	1850
Accuracy			0.99	3116
Macro avg	0.99	0.98	0.99	3116
Weighted avg	0.99	0.99	0.99	3116

where the training set is 75% of the available data and the testing set is the remaining 25%; Recall = TP/(TP + FN) (TP = True Positive, FN = False Negative, FP = False Positive, TN = True Negative); Precision = TP/(TP + FP); f1-score = 2*(precision*recall)/(precision + recall); accuracy = (TP + TN)/(TP + TN + FP + FN)

However, the MLP's are trained in batch mode and remain static after training, therefore the estimator is not adaptable to persistent changes in the link. Batch or offline training of ML algorithms (Banerjee and Basu, 2007) means that the model is trained, optimized and evaluated once on the training and test sets available and must be completely retrained later to accommodate possible changes in the file dissemination of updated data. In practice, this corresponds sporadic updates, for example, once every few hours and once for day depending on how the whole system was designed. For in the case of embedded devices, the device must be fully or partially reprogrammed (Ruckebusch *et al*., 2016). This consideration therefore prompted us to evaluate whether it was possible to add an unsupervised approach to the MLP so that the agent can self-learn the most suitable policy as the surrounding conditions change.

Reinforcement Learning Approach

Suppose our agent, to whom a central entity has sent a policy, is experiencing sudden interference. We consider, for example, the problem of radio channel selection. It will take some time before the new policy is recalculated and sent back to our agent. So, the objective of the agent is to choose for the next time slot a channel with no interference. Suppose the external interference has a periodic pattern, sweeping over all channels one to four in the same order. The agent must now autonomously learn a strategy that allows him to avoid the interfered time slots. In this case a Reinforcement Learning approach, in particular a Q-Learning Model, can be the solution. In this sense, our simulation environment transfers control to the agent, who autonomously identifies the appropriate policy for the new situation.

Q-Learning Model

In this case we have to take into account the protocol stack of our system, as learning, now, takes place in real time. To do this we can use ns-3. Ns-3 is a discrete-event network simulator for Internet systems (ns-3 project, 2020). In order to make ns-3 communicate with a Reinforcement Learning algorithm in OpenAI-gym we used ns3-gym. OpenAI Gym is a toolkit for Reinforcement Learning (RL) widely used in research. Ns3-gym is a framework that integrates both OpenAI Gym and ns-3 (Gawłowicz and Zubow, 2019).

Q-Learning is a model-free application of machine learning, that is the AI "agent" does not need to know the environment that it will be in. Indeed, the same algorithm can be used across different environments. Once defined the environment, everything is splitted into "states" and "actions."

The states are observations of the environment and the actions are the choices the agent has made based on the observation. Table 4 shows the RL mapping that has been used by Gawlowicz.

The agent doesn't really need to know anything about the environment. For each environment, the agent can query for how many actions are possible. In this case, there are "4" actions. When the agent steps the environment, it act with a 0, 1, 2 or 3 as its "action" for each step. Each time it does this, the environment will return to him the new state, a reward, whether the environment is done and then any extra info that some envs might have. A "0" means go to timeslot 1, 1 means go to TS 2 and so on. All the agent needs to know is what the options for actions are and given a state, what the reward of performing a chain of those actions would be. The agent knows he can take 4 actions at any given time. That's the "action space". Now, we need the "observation space." In this gym environment, the observations are returned from resets and steps. The "observation" is given by the information of which of the four time slots is interfered at that time.

The way Q-Learning works is based on a "Q" value per action possible per state. This produces a table. To figure out all of the possible states, the agent can either query the environment or just simply has to engage in the environment for a while to figure it out. It will check this table to determine the moves. When the agent is being "greedy" and trying to "exploit" its environment, it will choose to take the action that has the highest *Q* value for this state. However, sometimes, especially at the beginning, it may decide to "explore" and choose a random action. These random actions are the way our model will learn better moves over time. *Q* values are updated this way:

$$Q^{new}(s_t,a_t) \leftarrow (1-\alpha)\cdot Q(s_t,a_t) + \alpha\cdot\left(r_t + \gamma\cdot\max_a Q(s_{t+1},a)\right) \quad (28)$$

Where:

r_t	=	Reward
γ	=	Discount factor
$\max_a Q(s_{t+1}, a)$	=	Estimate of optimal funture value
α	=	Learning rate
$Q(s_t, a_t)$	=	old value

Table 4: Reinforcement learning mapping

Observation	Occupation on each channel in the current time slot
Actions	Set the channel to use for the next time slot
Reward	+1 in case of no collision with interferer; -1 otherwise
Gameover	If more than 3 collisions occur during the last ten time-slots

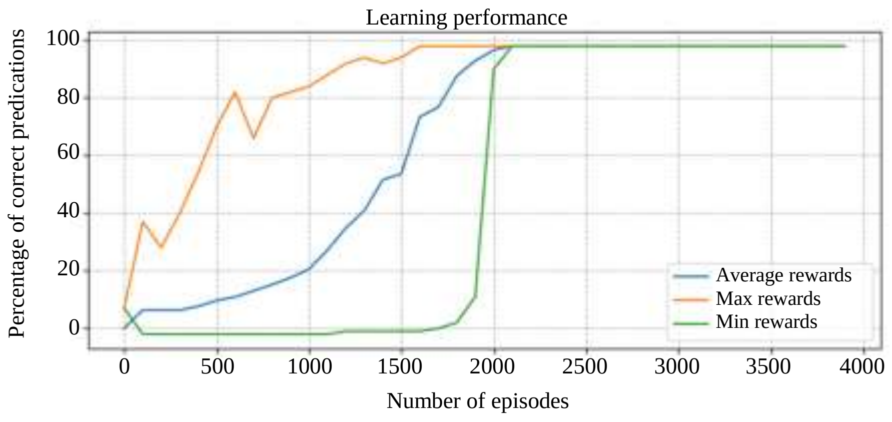


Fig. 11: Learning performance of Q-learning model

The Discount is a measure of how much the agent wants to care about future reward rather than about immediate reward. Typically, this value is between 0 and 1. The higher the better, because the purpose of Q Learning is, indeed, to learn a chain of events that ends with a positive outcome, so it's natural that the agent put greater importance on long terms gains rather than short term ones. The max_future_q is determined after the agent has performed its action already and then it updates its previous values based partially on the next-step's best Q value. Over time, once the agent has reached the objective, this "reward" value gets slowly backpropagated, one step at a time, per episode.

Results of Q-Learning Model

Figure 11 shows the learning performance using a modified version of the Q-Learning algorithm used by Gawlowicz. The modified version of the algorithm can be found in a GitHub Repository[1]. The main difference we introduced, compared to the original version, is related to the libraries used. We have eliminated the dependence on libraries such as Tensorflow and Keras. These libraries in fact, while ensuring high performance, use AVX instructions which may not run on older CPUs. In the original version we could see that after 80 episodes the agent will be able to perfectly predict the next channel state from the current observation so avoiding any collision with the interference. In our modified version we need some more episodes, about 600 episodes. On the other hand, the advantage is that the modified version can be used even on Cognitive Radio with limited computational power, such as NB-IoT, Sigfox and LoRaWan devices, because it does not require GPU support and high performing CPU, since in the prediction were not employed high performance numerical computation tools such as (Tensorflow, 2019; Keras, n.d.).

[1] https://github.com/apirodd/apirodd/projects?query=is%3Aopen

Table 5: Time complexity comparisons for RL algorithms on episodic MDP. T = KH is the total number of steps, H is the number of steps per episode, S is the number of states and A is the number of actions (source (Jin *et al*., 2018))

	Algorithm	Time	Space
Model-based	RLSVI	$\tilde{O}(TS^2A^2)$	$O(S^2A^2H)$
	UCRL2	$\Omega(TS^2A)$	$O(S^2AH)$
	Agrawal and Jia		
	UCBVI	$\tilde{O}(TS^2A)$	
	Vucq		
Model-free	Q-learnig (ε greedy) (if 0 initialized)	$O(T)$	$O(S^2AH)$
	Delayed Q-learning		
	Q-learning (UCB-H)		
	Q-learning (UCB-B)		
	Lower bound	-	-

Discussion

It has been shown in (Xu and Gu, 2020) that neural Q-learning with Multiple Layers finds the optimal policy with O(1/sqrt(T)) convergence rate if the neural function approximator is sufficiently overparameterized, where T is the number of iterations.

Table 5 from (Jin *et al*., 2018), shows that the Time complexity for the Model-free scenario is $O(T)$ where T is the total number of steps.

In real-time applications, the appropriate task representation or suitable initial Q-values is very important. In fact, prior results indicated that reinforcement learning algorithms are exponential in "*n*" (number of states), thus limiting their practical use if this set is high dimensional. In (Koenig and Simmons, 1993) has been shown that such algorithms are tractable if we use appropriate initial Q-values.

Further studies are moving towards the analysis of a multi-agent interaction (Multi Agent Reinforcement Learning-MARL). This would allow the different devices to cooperate by identifying a multi-agents policy, addressing the sequential decision-making problem when they are operating in a common environment. Each agent aims to optimize its own long-term reward by interacting with the environment and other agents (Busoniu *et al*., 2008), in particular, both the evolution of the system state and the return received by each agent are influenced by the joint actions of all agents (Zhang *et al*., 2019).

Conclusion

Over the next few years, the growth of mobile devices will grow steadily while the radio resource will remain substantially unchanged. It is therefore necessary to provide strategies for an optimized use of the radio channel. In this study we have shown a possible approach to face the problem, highlighting how the combined use of supervised learning and reinforcement learning models applied to predicting the behavior of the transmission channel can provide interesting results on the performance of the entire system.

Author’s Contributions

Andrea L. Piroddi: Designed the research plan, organized the study and participated in all experiments (In particular, he wrote and ran the Machine Learning Codes.), coordinated the data-analysis and contributed to the writing of the manuscript.

Maurizio Torregiani: Participated in all experiments, mainly contributing on the radio propagation aspects inside the paper. Verified the consistency of results of the experiments and contributed to the writing of the manuscript.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Andrews, J. G., Ganti, R. K., Haenggi, M., Jindal, N., & Weber, S. (2010). A primer on spatial modeling and analysis in wireless networks. IEEE Communications Magazine, 48(11), 156-163. https://doi.org/10.1109/MCOM.2010.5621983

Banerjee, A., & Basu, S. (2007, April). Topic models over text streams: A study of batch and online unsupervised learning. In Proceedings of the 2007 SIAM International Conference on Data Mining (pp. 431-436). Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611972771.40

Busoniu, L., Babuska, R., & De Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews), 38(2), 156-172. https://doi.org/10.1109/TSMCC.2007.913919

Cerar, G., Yetgin, H., Mohorčič, M., & Fortuna, C. (2018). Machine Learning for Wireless Link Quality Estimation: A Survey. arXiv preprint arXiv:1812.08856. https://doi.org/10.1109/COMST.2021.3053615

CISCO. (2020). Cisco Annual Internet Report (2018-2023) White Paper. CISCO.

Gawłowicz, P., & Zubow, A. (2019, November). Ns-3 meets OpenAI gym: The playground for machine learning in networking research. In Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (pp. 113-120). https://doi.org/10.1145/3345768.3355908

Gonzalez-Ruiz, A., Ghaffarkhah, A., & Mostofi, Y. (2011). A comprehensive overview and characterization of wireless channels for networked robotic and control systems. Journal of Robotics, 2011. https://doi.org/10.1155/2011/340372

Jin, C., Allen-Zhu, Z., Bubeck, S., & Jordan, M. I. (2018). Is Q-learning provably efficient?. arXiv preprint arXiv:1807.03765. https://arxiv.org/abs/1807.03765

Keras. (n.d.). Keras: The Python Deep Learning library. https://keras.io/

Koenig, S., & Simmons, R. G. (1993, July). Complexity analysis of real-time reinforcement learning. In AAAI (pp. 99-107). http://www.ri.cmu.edu/pub_files/pub1/koenig_sven_1993_1/koenig_sven_1993_1.pdf

Nordin, R. (2012). An Investigation of Self-Interference Reduction Strategy in a Spatially Correlated MIMO Channel. Journal of Computer Networks and Communications, 2012. https://doi.org/10.1155/2012/424037

ns-3 project. (2020). ns-3 Tutorial. https://www.nsnam.org/docs/tutorial/html/

Pandas Devel. Team. (2020). Pandas Documentation. https://pandas.pydata.org/docs/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830. https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf?source=post_page---------------------------

Publications Office of the EU. (1999). COST Action 231. Publications Office of the EU. https://op.europa.eu/en/publication-detail/-/publication/f2f42003-4028-4496-af95-beaa38fd475f

Rondeau, T. W., & Bostian, C. W. (2009). Artificial intelligence in wireless communications. Artech House.

Ruckebusch, P., De Poorter, E., Fortuna, C., & Moerman, I. (2016). Gitar: Generic extension for internet-of-things architectures enabling dynamic updates of network and application modules. Ad Hoc Networks, 36, 127-151. https://doi.org/10.1016/j.adhoc.2015.05.017

Samek, W., Stanczak, S., & Wiegand, T. (2017). The convergence of machine learning and communications. arXiv preprint arXiv:1708.08299. https://arxiv.org/abs/1708.08299

TensorFlow. (2019). TensorFlow: An open source machine learning framework for everyone. https://www.welcome.ai/tech/deep-learning/tensorflow

Xu, P., & Gu, Q. (2020, November). A finite-time analysis of Q-learning with neural network function approximation. In International Conference on Machine Learning (pp. 10555-10565). PMLR. http://proceedings.mlr.press/v119/xu20c.html

Zhang, K., Yang, Z., & Başar, T. (2019). Multi-agent reinforcement learning: A selective overview of theories and algorithms. arXiv preprint arXiv:1911.10635. https://arxiv.org/abs/1911.10635