Deep Neural Network Parameterization for Channel Estimation in MUSA Systems

Mariana Baracat de Mello and Luciano Leonel Mendes

***Abstract*— Multi-user shared access is a non-orthogonal multiple access technique that has been considered as a potential solution for 5G and beyond wireless networks. However, its performance is affected by the propagation of the channel estimate error in the SIC algorithm in the multi-user detector. Deep neural networks can be used to improve the initial channel estimate and this paper analyzes how the adjustment of hyperparameters and randomness affect the performance of the proposed estimator.**

***Keywords*— hyperparameters, MUSA, deep learning, NOMA.**

I. Introduction

The applications foreseen for massive machine type communications (mMTC) scenario require high spectrum and energy efficiency and non-orthogonal multiple access (NOMA) techniques are possible candidates to meet these requirements [1]. Unlike traditional orthogonal multiple access (OMA) schemes, NOMA allows signals from multiple users to share the same time-frequency resource via superposition at the cost of higher complexity at the receiver.

Multi-user shared access (MUSA) is a NOMA technique designed for the mMTC scenario, which can support a large number of connections and, at the same time, minimize signaling overhead and energy consumption [2]. MUSA is based on grant-free access and employs non-orthogonal short-length complex sequences to spread user data. Since no pilot is used in the frame structure, the receiver performs blind detection based on successive interference cancellation (SIC) to obtain the channel state information (CSI).

CSI has a high impact on the performance of the MUSA system since poor channel estimation reduces the capability of the SIC to remove the multi-user interference from the received signal [3]. Deep learning (DL) is a powerful tool for solving nonlinear problems and can be used to improve a channel estimate. In this paper, a deep neural network (DNN) is trained with a set of channel gains estimated by least squares (LS) under high signal-to-noise ratio (SNR) and with a low overload factor, as proposed in [4]. Once trained, the DNN can be used to improve the channel estimation for different overload factors.

One of the biggest challenges when working with neural networks is to define what is the best configuration of the model [5]. Hyperparameters are configurations of the machine learning (ML) model that can be adjusted to improve the performance and quality of the learning algorithm [6]. The main goal of this paper is to show how the hyperparameters adjustment and the randomness present in neural networks can affect the performance of the channel estimation.

The paper is organized as follows. Section II presents the principles of the MUSA scheme and blind detection based on SIC. Section III presents the DNN employed to improve channel estimation and how the hyperparameters affect DNN performance. Section IV presents the evaluation of numerical results for different DNN parameterizations. Finally, the conclusions are provided in Section V.

M. B. de Mello and L. L. Mendes are with Instituto Nacional de Telecomunicações (Inatel), Santa Rita do Sapucaí, MG, 37540-000, Brazil (email: marianam@gee.inatel.br and luciano@inatel.br).

II. Principles of MUSA

MUSA is a NOMA scheme that employs complex spreading sequences to accommodate a large number of users sharing a limited number of time-frequency resources. A ternary short complex sequence is used to spread the users information. The codes can be obtained from two-dimensional constellations, where the center of complex plane is included in the basis sequence. For example, based on the rectangular 8-QAM constellation, the elements of the complex sequence are $\{0, 1, 1+i, i, -1+i, -1, -1-i, -i, 1-i\}$ [2]. Therefore, 9^L complex spreading sequences of length L can be obtained from this alphabet.

A codebook $\mathbf{G}$ $(L \times Q)$ is generated with Q pre-selected spreading sequences. This codebook is available to users to spread the data and also to the base station (BS) receiver. Since the users choose the sequence autonomously, i.e., grant free access, two or more users can choose the same spreading sequence, resulting in a collision. This problem can be minimized by increasing Q, at the cost of higher complexity at the BS. The relatively low cross-correlation between spreading sequences can reduce the interference among users and, improving the detection robustness [2].

On the transmitter side, the vector of data bits $\mathbf{b}_k$ $(U \times 1)$ generated by the kth user is encoded by a channel encoder with code rate $R = \frac{U}{V}$. The coded bits $\mathbf{c}_k$ $(V \times 1)$ are modulated by a M-QAM modulator, where M is the order of the constellation, resulting in the modulated symbols vector $\mathbf{m}_k$ $(J \times 1)$, with $J = \frac{V}{log_2(M)}$. The vector $\mathbf{m}_k$ is spread over the complex spreading sequence $\mathbf{s}_k$ $(L \times 1)$, chosen randomly from $\mathbf{G}$ by the user. Finally, the spreading symbols $\mathbf{x}_k$ $(LJ \times 1)$ are transmitted using the time-frequency resources shared by users, e.g., orthogonal frequency division multiplexing (OFDM) subcarriers [7]. For this, $N < LJ$ OFDM subcarriers are allocated using the inverse fast Fourier transform (IFFT) and $\frac{LJ}{N}$ OFDM symbols are required for the transmission of $\mathbf{x}_k$. Thus, the OFDM frame for the kth user is represented by $\mathbf{X}_k$ $\left(N \times \frac{LJ}{N}\right)$.

On the receiver side, the SIC algorithm is used to decode the K users' data bits of the received superimposed signal $\mathbf{Y}$

$\left(N \times \frac{LJ}{N}\right)$. Assuming that the entire OFDM frame has been received at the BS, the signal on the nth subcarrier of the ith OFDM symbol is given by [4]

$$y^{n,i} = \sum_{k=1}^{K} x_k^{n,i} h_k^{n,i} + w^{n,i}, \tag{1}$$

where $n = (1, \cdots, N)$, $i = (1, \cdots, \frac{LJ}{N})$, $x_k^{n,i}$ is the element of the nth row and ith column of the matrix $\mathbf{X}_k$, $h_k^{n,i}$ is the channel response between the kth user and the BS and $w^{n,i}$ is the additive white Gaussian noise (AWGN).

SIC algorithm is used on the BS side to remove the multi-user interference. The SIC performance improves when the SNR varies due to the near-far effect [2]. The SIC uses the information retrieved from a reliable link, i.e., a higher SNR, to remove the interference introduced in the data transmitted by other users with a lower SNR. Therefore, this algorithm is executed K times, starting from the user signal with the highest SNR to the user signal with the lowest SNR.

Initially, the data from the user with the highest SNR must be retrieved correctly. Thus, (1) can be rewritten as

$$y^{n,i} = x_\varphi^{n,i} h_\varphi^{n,i} + \sum_{\substack{k=1,\\ k\neq\varphi}}^{K} x_k^{n,i} h_k^{n,i} + w^{n,i} \tag{2}$$

where $x_\varphi^{n,i}$ represents the data transmitted by the user with highest SNR at the nth subcarrier of the ith OFDM block and $\sum_{k=1,\, k\neq\varphi}^{K} x_k^{n,i} h_k^{n,i}$ is the multi-user interference. The first detection step is to decouple the OFDM subcarriers by applying the fast Fourier transform (FFT) and organize the result into a vector $\hat{\mathbf{x}}_\varphi$. The cross-correlation between $\hat{\mathbf{x}}_\varphi$ and all sequences of $\mathbf{G}$ is performed to find the spreading sequence used on the transmit side. Then, the Q versions of the despread signal $\hat{\mathbf{m}}_\varphi$ are detected, resulting in vectors of coded bits $\hat{\mathbf{c}}_\varphi$. After demodulation, each version of $\hat{\mathbf{c}}_\varphi$ is decoded by the forward error correction (FEC), and the cyclic redundancy check (CRC) is verified. The spreading sequence that results in error-free data is considered the chosen sequence by the user during transmission. Therefore, the received SNR must be high enough to assure the proper recovery of the transmitted bits $\hat{\mathbf{b}}_\varphi$.

Assuming that $\mathbf{s}_\varphi$ is the sequence used on the transmission side, the vector $\hat{\mathbf{b}}_\varphi$ is encoded, modulated, spread, and the IFFT is applied to generate the feedback matrix $\hat{\mathbf{X}}_\varphi$. This signal is considered equal to that transmitted by the user with highest SNR and must be subtracted from the signal received to cancel its interference on the other users. To perform the subtraction is necessary to weigh $\hat{\mathbf{X}}_\varphi$ by the channel response. As the CSI is not known, the channel gain can be estimated from $\hat{\mathbf{X}}_\varphi$ and $\mathbf{Y}$. Assuming the LS estimation, the estimated channel response is given by [4]

$$\hat{\mathbf{h}}^i = \left(\hat{\mathbf{x}}^{i\mathrm{H}} \hat{\mathbf{x}}^i\right)^{-1} \hat{\mathbf{x}}^{i\mathrm{H}} \mathbf{y}^i, \tag{3}$$

where $i = 1, 2, \cdots, \frac{LJ}{N}$, $(\cdot)^{\mathrm{H}}$ is the Hermitian matrix, and $\hat{\mathbf{x}}^i$ and $\mathbf{y}^i$ are vectors $(N \times 1)$ formed by the ith column of $\hat{\mathbf{X}}_\varphi$ and $\mathbf{Y}$, respectively. The LS estimator provides a poor channel estimation because of the noise [4]. Thus, the error in $\hat{\mathbf{H}}$ $\left(N \times \frac{LJ}{N}\right)$ propagates through the SIC receiver, reducing its performance.

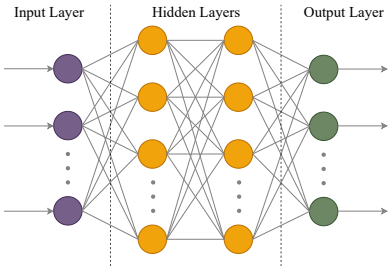

Fig. 1. Deep neural network structure.

III. DNN for Channel Estimation

The DNN as a non-linear processing unit can improve the channel estimation. Due to the superior ability to approximate abstract and non-linear functions, DNN is applied on top of the LS estimate to reduce error on of LS estimate. DNNs are neural networks with one or more hidden layers to improve the representation capacity, as shown in Fig. 1. The layers are composed of several processing units called neurons, which connect to the neurons of the next layer. Each neuron has an output that is a nonlinear activation function of a weighted sum of neurons from the previous layer. Assuming a DNN with D layers and A neurons per layer, the output of a neuron is given by

$$\tilde{z}_a^{(d)} = f\left(\tilde{\mathbf{w}}_a^{(d)\mathrm{T}} \tilde{\mathbf{z}}^{(d-1)} + \tilde{b}_a^{(d)}\right), \tag{4}$$

where $(.)^{\mathrm{T}}$ is the transpose, $d = (1, \cdots, D)$ is the layer index, $a = (1, \cdots, A)$ is the neuron index in layer d, $f(.)$ is the activation function, $\tilde{b}_a^{(d)}$ is the bias, $\tilde{\mathbf{w}}_a^{(d)}$ is the weight vector, and $\tilde{\mathbf{z}}^{(d-1)}$ is the input vector given by the output of the previous layer. Thus, $\tilde{\mathbf{z}}^{(0)}$ is the input vector, given by the concatenated real and imaginary parts of the LS estimate. The DNN output is a nonlinear cascade transformation of the input data.

This paper considers the training of the DNN, where the weights of the neural network are updated iteratively to minimize a specific loss function, such as the mean squared error (MSE) between the estimated value at its output and the actual value. Thus, the MSE loss function can be expressed as

$$\epsilon = \frac{1}{N_t} \sum_{i=1}^{N_t} \left\| \tilde{z}_{\mathrm{D}}^T - \tilde{z}_{\mathrm{D}}^P \right\|^2, \tag{5}$$

where $\tilde{z}_{\mathrm{D}}^P$ is the channel estimated by DNN, $\tilde{z}_{\mathrm{D}}^T$ is the true channel and N_t is the number of training samples. The DNN should be trained from known pairs of input and output data, allowing it to learn the weight vector. One of the problems faced in the training of neural networks is the definition of its hyperparameters. Small differences in the adjustment of these hyperparameters can lead to substantial differences in training time and the generalization obtained. Also, ML involves storing and organizing parameters and results, ensuring that they are reproducible.

A. Neural Network Fine-tuning Hyperparameters

In machine learning, most algorithms have hyperparameters that can be adjusted to control the algorithm's behavior [5]. However, there is a difference between the parameters and hyperparameters of a learning model. The model parameters can be defined as internal configuration variables, which can be estimated from the data. An example of a parameter is the weights of neural networks that have their values derived from training. Hyperparameters are variables whose values are adjusted to control the learning process. In other words, hyperparameters are the variables that determine the model's structure and how it is trained, for example, the number of hidden layers and learning rate. Therefore, the choice of hyperparameters affects the speed and quality of the learning process.

Tools such as Grid Search can assist in adjusting the model's hyperparameters. However, DNN can have many hyperparameters to be adjusted, and training for a large data set takes a long time. This means that only a part of the hyperparameter space can be explored within an acceptable time frame. Therefore, it is essential to be aware of what values would be reasonable for the initialization of each hyperparameter to restrict the search space [8].

Next, the main hyperparameters of a neural network and the good practices that can help the choice of these configurations will be presented.

1) Number of Hidden Layers: In many problems, it is possible to obtain acceptable results with a single hidden layer, as long as there are enough neurons. However, it is fundamental to note that neural networks with many hidden layers have a much higher parameter efficiency than neural networks with few layers. They use exponentially fewer neurons than low-layer networks to model complex functions, making training much faster. Also, computational models composed of many hidden layers learn to represent data at multiple levels of abstraction.

So for many problems, only one or two hidden layers work very well [8]. While for more complex tasks, the number of hidden layers can be increased gradually until the model starts overfitting and the training data set must be large enough. However, it is common to reuse parts of a pre-trained network that performs a similar task. This makes training faster and requires a much smaller data set [8].

2) Number of Neurons in the Hidden Layers: The number of neurons in the input and output layers depends on the input data and the type of output the task requires. Regarding hidden layers, it is important not to use too many neurons, as this causes the model to memorize the training data and lose its ability to generalize, causing overfitting. Neither should neurons be used too little, as it ends up forcing the network to spend too much time trying to find an optimal representation, which can result in underfitting.

A common practice is to use the same number of neurons in all hidden layers, this makes it necessary to adjust only one hyperparameter instead of one per layer. This allows to gradually increase the number of neurons in all layers until the model starts overfitting. Another approach would be to configure an overfitted model with more layers and numbers of neurons than necessary and, then, use the early stopping technique of training. Unfortunately, this increases the computational costs associated with the model. In summary, many neurons in a hidden layer with regularization techniques can increase accuracy. Fewer neurons can result in underfitting.

3) Learning Rate: According to the results, the learning rate is perhaps the most difficult hyperparameter to adjust. Since if the learning rate is too low, the learning of the neural network becomes very slow. And if the learning rate is too high, it causes fluctuations in training and prevents the convergence of the learning process. Therefore, an inadequate learning rate, whether too large or too low, results in a model with low effective capacity due to optimization failure [5]. In the last few years, optimizers faster than the Gradient Descent have been used to accelerate the model training. These optimizers are adaptive learning rate algorithms and, therefore, require fewer adjustments to the learning rate, making it easier to use than the Gradient Descent. The main adaptive learning rate optimizers are AdaGrad, RMSProp, and Adam [8].

B. Randomness in Neural Networks

In addition to adjusting hyperparameters, it is necessary to ensure that the results and the model are reproducible. The lack of compatibility and reproducibility of a model makes it very difficult to correctly interpret the results, especially for other researchers.

Machine learning algorithms often exhibit stochasticity inherent in learning, which can be induced by the randomness present, for example, in data collection, in the order in which the examples are exposed to the model, in the initialization of weights in an DNN and resampling approaches. Randomness is an important part of many machine learning algorithms, however it is an obstacle to reproducibility.

Reproducibility is a necessary practice in situations where it is desired to compare different techniques and algorithms. For machine learning models to be reproduced, it is necessary to have the same code, data set, and sequence of random numbers. The solution is to use pseudo-random number generators that take a seed and generate seemingly random numbers in a deterministic way. In addition, if the seed is defined with a fixed value, all executions will share the same randomness, making possible the exact reproduction of the model.

IV. Numerical Results

The MUSA system used in the simulations was generated as described in Section II with the parameters shown in Table I. The differential binary phase shift keying (DBPSK) modulation was chosen because it allows for phase tracking on the receiver side. As the CSI is not previously known, the DBPSK modulator inserts a reference symbol in the modulated symbol block for phase tracking. In addition, a polar code-based FEC was employed.

To improve the initial LS estimate given in (3), it was proposed to use a DNN to correct the general estimate error. Thus, the proposed DNN is fed with the LS channel estimates and on its output, the improved LS channel response estimates are obtained. As the LS estimate is complex, it is necessary

TABLE I
CONFIGURATION PARAMETERS OF THE MUSA SYSTEM.

Parameters of the MUSA System	
Block size of data bits (U)	64
Block size of encoded bits (V)	191
Block size of modulated symbols (J)	192
Spreading sequence length (L)	4
Number of subcarriers per OFDM block (N)	12
Number of pre-selected spreading sequences (Q)	64

to transform the LS estimates from complex values to real values by stacking the real and imaginary parts. Thus, the DNN input and output layers have a fixed number of neurons equal to $2N$. The DNN output values are converted back to the complex values.

The data set used for training has 10000 samples generated considering SNR equals 40 dB, divided into 8000 training samples and 2000 validation samples. A high SNR was used for the training to reduce the impact of the noise, which allows DNN to learn more about the characteristics of the channel. To evaluate the performance of the proposed estimator, the trained DNN was tested with a set of test data with different SNR values in the set $\{4, 11.2, 18.4, 25.6, 32.8$ and $40\}$ dB. The dataset used in the test to evaluate the generalization of the trained neural network is different from the dataset used during the training and validation process. Since supervised learning is used to train the model, the training set includes the LS estimates of the channel response and the corresponding true channel responses. The MSE was used as a loss function during training and a performance metric during the testing process.

The performance of the DNN was evaluated under the following hyperparameter adjustments:number of hidden layers, number of neurons per hidden layer and, learning rate. The hyperparameters used in Figs. 2, 3, 4, and 5 are shown in Table II. The other configurations, which do not change in the DNNs simulated in this section, are shown in Table III. In addition, due to space restrictions, the loss function curves were omitted during the training stage since the training and test curves converged to low loss values, and overfitting was not observed in any of the simulations presented in this article.

Due to the randomness present in the DNNs, as mentioned in Section III, seeds for the random number generator were defined in NumPy and Tensorflow, making the models reproducible and facilitating the comparison of the adjustment effect of each hyperparameter. In addition, the average performance is calculated through the random repetition of the learning algorithm for different proposed DNN architectures. In Fig. 2, 3, and 4, performance is evaluated considering the two values for the seed and the average of 30 random runs of the learning algorithm, i.e., the training and testing process of the neural network is executed 30 times without values fixed for the seeds. It will guarantee at each execution of the learning algorithm that the training and validation sets are different and that the weights are initialized randomly.

Fig. 2 shows the performance of the DNN estimator in terms of MSE under different SNRs and for DNN architectures with

TABLE II
CONFIGURATION OF HYPERPARAMETERS FOR THE DNNS PROPOSED IN FIGS. 2, 3, 4 AND 5.

	Hyperparameter	DNN 1	DNN 2	DNN 3
Fig. 2	Hidden layers	1	2	3
	Neurons per hidden layer	24	24	24
	Learning rate	0.001	0.001	0.001
Fig. 3	Hidden layers	1	1	1
	Neurons per hidden layer	12	24	32
	Learning rate	0.001	0.001	0.001
Fig. 4	Hidden layers	1	1	-
	Neurons per hidden layer	24	24	-
	Learning rate	0.001	0.01	-
Fig. 5	Hidden layers	1	2	2
	Neurons per hidden layer	24	12	12
	Learning rate	0.001	0.001	0.01

TABLE III
GENERAL CONFIGURATION OF PARAMETERS FOR THE DNNS PROPOSED IN FIGS. 2, 3, 4, AND 5.

Base DNN Parameters	
Activation function	ReLU
Optimizer	ADAM
Number of epochs	500
Batch size	128
Training samples	8000
Testing samples	2000

1, 2, and 3 hidden layers. It is noticeable that the performance of the estimate improves as the number of hidden layers increases. It is also possible to observe that the performance of the DNN estimator exceeds the performance of the LS estimate in all proposed hidden layer configurations. Finally, Fig. 2 shows that the average performance is close for 2 and 3 hidden layers, and, therefore, 2 hidden layers can be a good fit since the computational complexity is less.

Fig. 3 shows the performance of the DNN estimator in terms of MSE under different SNR for DNN architectures, with a hidden layer and different amounts of neurons in the hidden layer 12, 24, and 32, as shown in Table II. It is possible to observe the effect caused in the estimation by randomness since the number of neurons unevenly affects the performance of each seed. Also, it is possible to observe that the average performance is very close for the three quantities of neurons in the hidden layer. However, with 12 neurons, the generalization capacity of the trained model is slightly higher.

The performance of the DNN estimator for learning rates equal to 0.001 and 0.01 is shown in Fig. 4. For the configurations proposed in Table II, the performance of the DNN estimate is better for a learning rate equal to 0.01. And, as in Fig. 2 and Fig. 3, it is possible to notice that the performance of the DNN estimate is better than that of the LS estimate.

Finally, Fig. 5 shows the average performances of 30 random runs of the learning algorithm for three different DNN architectures. These are three possibilities for adjusting the hyperparameters that result in good performance of the DNN estimate with low training complexity. For two hidden layers, with 12 neurons per hidden layer and a learning rate equal

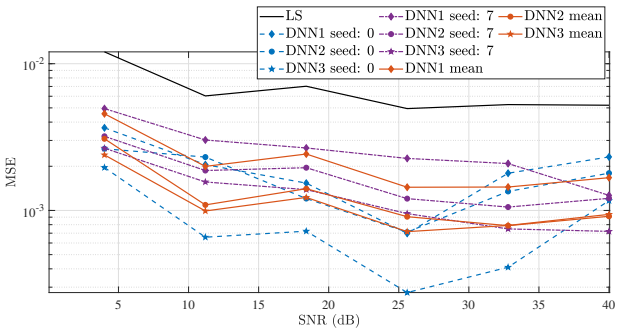

Fig. 2. Performance of the DNN based estimator in the online implementation stage, under different adjustments in the number of hidden layers.

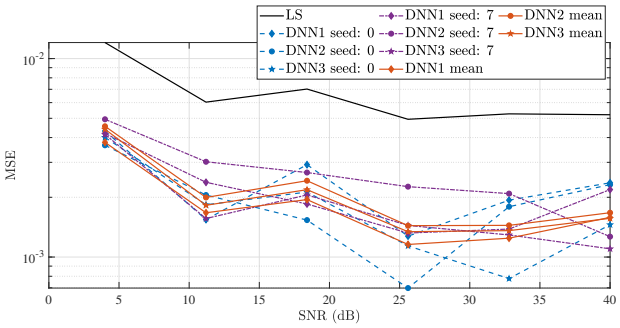

Fig. 3. Performance of the DNN based estimator in the online implementation stage, under different adjustments in the number of neurons per hidden layer.

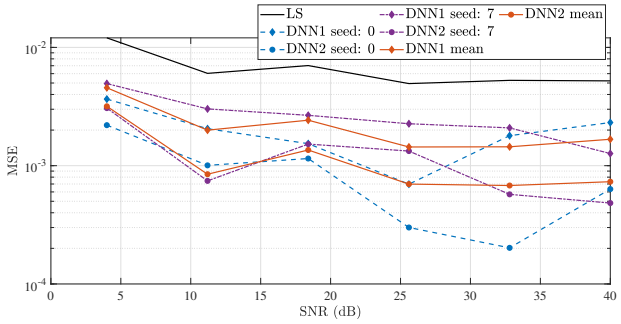

Fig. 4. Performance of the DNN based estimator in the online implementation stage, under different adjustments in the learning rate.

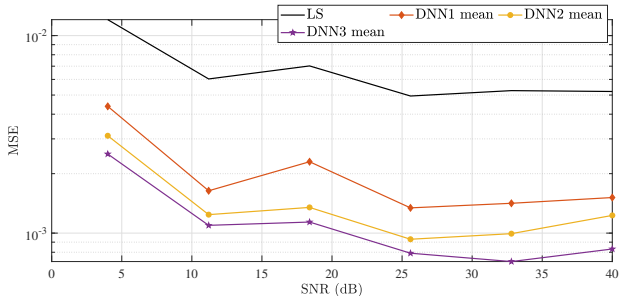

Fig. 5. Performance of the DNN based estimator in the online implementation stage, under different DNN architectures.

to 0.01, the generalization capacity of the DNN estimator is higher, and the performance in this scenario is better.

Through the results, it is possible to see that the performance of a neural network can change depending on the seed value. In Figs. 2, 3, and 4, the performance deteriorates for higher SNR values when the seed value is 0, while for seed equal to 7, the performance improves at higher SNR values. Therefore, for a more accurate analysis of the results, the learning algorithm must be executed a few times, and the average of these executions must be taken into account.

It is important to note that the results are generally specific to each task and cannot be transferred from one problem to another. However, there are broader trends in the adjustment of hyperparameters for each task that must be taken into account to facilitate and delimit the search space. The proper adjustment of hyperparameters leads to good performance results at the cost of appropriate complexity for the task in question, in addition to avoiding overfitting.

V. Conclusions

The DNN-based estimator can be used to improve the LS channel estimation and, consequently, improve the performance of the MUSA scheme. Adjusting the DNN hyperparameters is essential for improving the channel estimation, increasing the model's generalization, and reducing the complexity and training time. In addition, knowing how to initialize hyperparameters correctly can be useful to restrict the search space when using optimization tools, such as grid Search, and when training is very expensive due to the size of the data set and of the DNN structure. Finally, the stochastic nature of the learning algorithm must be taken into account when evaluating the model's performance and when reproducing it.

Acknowledgments

This work was partially supported by the MAI/DAI project (403612/2020-9), Brasil 6G project (01245.010604/2020-14), RNP, CPqD and CNPq-Brasil.

References

[1] Z. Wei, J. Yuan, D. W. K. Ng, M. Elkashlan, and Z. Ding, "A survey of downlink non-orthogonal multiple access for 5g wireless communication networks," 09 2016.

[2] Z. Yuan, G. Yu, W. Li, Y. Yuan, X. Wang, and J. Xu, "Multi-user shared access for internet of things," in *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, 2016, pp. 1–5.

[3] G. Gui, H. Huang, Y. Song, and H. Sari, "Deep learning for an effective nonorthogonal multiple access scheme," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 9, pp. 8440–8450, 2018.

[4] G. Aquino, T. Barbosa, M. Chafii, L. Mendes, and A. Gizzini, "MUSA grant-free access framework and blind detection receiver," *Journal of Communication and Information Systems*, vol. 36, no. 1, pp. 119–127, Jul. 2021. [Online]. Available: https://jcis.sbrt.org.br/jcis/article/view/771

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ser. Adaptive Computation and Machine Learning series. MIT Press, 2016. [Online]. Available: https://books.google.com.br/books?id=Np9SDQAAQBAJ

[6] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012. [Online]. Available: http://jmlr.org/papers/v13/bergstra12a.html

[7] E. Çatak, F. Tekçe, O. Dizdar, and L. Durak-Ata, "Multi-user shared access in massive machine-type communication systems via superimposed waveforms," *Physical Communication*, vol. 37, p. 100896, 2019.

[8] A. Géron, *Mãos à Obra: Aprendizado de Máquina com Scikit-Learn & TensorFlow*. Alta Books, 2019. [Online]. Available: https://books.google.com.br/books?id=Z0mvDwAAQBAJ