Learning resource allocation algorithms for cellular networks

Thi Thuy Nga Nguyen, Olivier Brun, Balakrishna Prabhu

▶ **To cite this version:**

Thi Thuy Nga Nguyen, Olivier Brun, Balakrishna Prabhu. Learning resource allocation algorithms for cellular networks. Machine Learning for Networking: Third International Conference (MLN 2020), Nov 2020, Paris, France. 10.1007/978-3-030-70866-5_12 . hal-03753556

HAL Id: hal-03753556
https://laas.hal.science/hal-03753556

Submitted on 18 Aug 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Learning resource allocation algorithms for cellular networks

Thi Thuy Nga Nguyen [†,⋆], Olivier Brun[⋆] and Balakrishna J. Prabhu[⋆]

[†] Continental Digital Service in France,Toulouse, France
[⋆] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

August 18, 2022

Abstract

Resource allocation algorithms in wireless networks can require solving complex optimization problems at every decision epoch. For large scale networks, when decisions need to be taken on time scales of milliseconds, using standard convex optimization solvers for computing the optimum can be a time-consuming affair that may impair real-time decision making. In this paper, we propose to use Deep Feedforward Neural Networks (DFNN) for learning the relation between inputs and the outputs of two such resource allocation algorithms that were proposed in [18, 19]. On numerical examples with realistic mobility patterns, we show that the learning algorithm yields an approximate yet satisfactory solution with much less computation time.

1 Introduction

In cellular wireless networks, a central scheduling problem is to choose one among several concurrent users (for example, mobile phones) to which the scheduler (henceforth also referred to as the base station) must send data to. This scheduling decision is taken every time-slot which is of the order of 2 ms [16] and is based on what are called as channel conditions of the users. Roughly, the channel condition of a user determines the data rate at which the base station can communicate with this user. In wireless networks, these conditions can vary randomly on short as well as on long time scales. Also called fading and shadowing, these random variations are a consequence of the interference patterns induced by the different obstacles (building, trees, etc.) in the path of the radio waves used for wireless communications [23]. However, the decision is not as easy as scheduling the user with the best channel condition as such a policy could lead to unfairness between users. Imagine a user with a direct line of sight path with the base station and

another who is inside a building or an underground metro station. Quite possibly, the former user will always have a better channel which would starve the latter user of any communication.

To avoid these unfair allocations, a typical solution is to define a utility which is a usually a concave function of the throughput[1] of the users and then compute the scheduling decision as the one that maximizes the sum of the utilities of the users. For example, the utility function could be the logarithm of the average data rates of the users. These solutions fall under the umbrella of the network utility maximization problem [27].

In an ideal scenario, the base station will solve this utility maximization problem every 2 ms. However, there are two practical issues that make this infeasible. First, the throughput of a user depends upon future channel conditions which are unknown to the scheduler. Second, the integer scheduling problem is known to be NP-complete [16]. Solving such an optimization problem over a time horizon of seconds (thousands of time-slots) and with hundreds of users can be unrealistic in time-slots of 2 ms (which are actually becoming shorter as the technology progresses). To overcome these practical issues, several heuristics have been proposed that are based on an estimation of the future data rates [16, 18, 19]. The heuristics (called STO1 and STO2) in [18, 19] use the estimated future data rates as an input to a relaxed version of the original problem restricted to a shorter time horizon thereby reducing the dimensionality of the problem. These heuristics are thus much faster to solve than the original problem but they still require solving rather frequently a large-scale concave optimization problem which can be time consuming. It was shown numerically in [18, 19] that STO1 and STO2 performed better than the one in [16] as well as the popular PF algorithm [13] that does not use future estimated rates. Therefore, in this paper, we shall address the problem of improving the speed of these heuristics without compromising on their superior total utility.

1.1 Contributions

We propose a machine learning based solution to speed up the operations of STO1[2]. The key idea is to use a Deep Feedforward Neural Network (DFNN) to approximate the output of the relaxed optimization problem in the heuristics. It will be shown on numerical examples that once the DFNN is trained, it takes much less time to generate a reasonable accurate solution compared to using the specialized Python package CVXPY [4] that uses

[1]We use data rate and the throughput to denote two different but related quantities. The throughput only takes into account the data rate of the time-slots in which a user is served. It is thus no more than the total data rate of this user.

[2]STO2 is similar to STO1 but solves the optimization problem less frequently. In this paper we shall focus on STO1 but the ideas developed here can be applied to STO2 as well.

the solver MOSEK [1] for solving of a concave optimization problem. We compare different DFNN architectures and different loss functions to find the most appropriate ones for our problem. We then compare the behavior of the learning algorithm with STO1 and with other existing algorithms as well. The comparison shall be done on scenarios created by SUMO [15] which can generate realistic mobility patterns on road networks with vehicular mobility. Based on numerical results, the learning algorithm is shown to perform close to STO1 with much less computation time.

1.2 Related works

The theory of approximation for DFNN has been studied in many papers. Motivated by Komogorov's superposition theorem [12] in 1957, many approximation results have proven the approximation capabilities of feed-forward neural networks for the class of continuous functions such as [3],[8],[17]. In his theorem, Komogorov proved that any continuous function can be represented as a superposition of continuous functions of one variable. In [3] (1989), Cybenko proved that any multivariate continuous function with support in a hypercube can be uniformly approximated by a linear finite combinations of compositions of a sigmoidal functions and a set of affine functions. This representation is in fact a feed-forward neural networks with sigmoidal activation functions. Independently with the work of Cybenko, Hornik [8] (1989) also proved a similar result. Two years later, Hornik [7] showed that multi-layer feed-forward neural networks with arbitrary bounded and non-constant activation function can approximate arbitrary well real-valued continuous functions on compact subsets of $\mathbf{R}^n$ as long as sufficiently many hidden layers are available. The word "deep" in "deep learning" thus simply means many layers.

Learning an algorithm to produce an approximate algorithm in order to reduce computation time has been proposed in several recent research papers [6], [21]. In [6], the authors consider a Sparse Coding problem which is used for extracting features from raw data. The problem is that Sparse Coding is often too slow for real-time processing in several applications such as pattern recognition. The authors propose a method using a non linear, feed-forward function to learn Sparse Coding to produce an approximate algorithm with 10 times less computation.

Learning an algorithm for wireless resource management has been proposed in [21]. In that work, the authors used DFNN to learn an algorithm for the interference channel power control problem. They obtain an almost real time algorithm, since passing the input through a DFNN to get the output only requires a small number of simple operations as compared to an iterative optimization algorithm. They show that, by choosing an appropriate initialization, the initial power control algorithm performs a continuous mapping which can be efficiently learnt.

In this paper, we use DFNNs for learning a channel allocation algorithm maximizing the proportional fairness between vehicular users. The proposed method is however potentially applicable to other convex optimization problems.

1.3 Organization

In Section 2, we recall the resource allocation problem in the case of a single Base Station (BS). We also remind the reader of the STO1 algorithm and state the learning problem we address in this paper. In Section 3, we formally define the input-output relationship for the DFNN model. Numerical results are presented in Section 4. We compare the computing times of the DFNN-based prediction algorithm against those of the original algorithm in Section 5 to evaluate the reduction in computing times. Finally, in Section 6 we discuss several research directions that can be followed in future work.

2 Problem Formulation

Consider the following downlink discrete-time channel allocation problem for a single Base Station (BS) (see [13] and references therein):

$$\left\{\begin{array}{ll} \text{maximize} & O(\alpha)=\sum_{i=1}^{K}\log\left(\sum_{j=1}^{T}\alpha_{ij}r_{ij}\right) \\ \text{subject to} & \\ & \sum_{i=1}^{K}\alpha_{ij}\leq 1, \quad j=1,\ldots,T; \\ & \alpha_{ij}\in\{0,1\}, \quad j=1,\ldots,T,\ i=1,\ldots,K. \end{array}\right. \tag{I}$$

Here $r_{ij}\geq 0$ is the data rate of user i in time-slot j, and α_{ij} is the corresponding allocation in this slot. The constraints impose that the BS can choose at most one user in each time-slot. The objective of the BS is to maximize the sum of the individual user utilities which are defined as the logarithm of the total throughput of the user over a time horizon of T. As mentioned in Section 1, solving this problem is not practical because the data rates $r_{i,j}$ become known to the scheduler only in slot j. Further, users arrive and leave and it is not possible to know in advance which users will be present in the network in the future. Hence, the algorithms proposed use either no information on the future rate (see [13] and references therein) or an estimation of the future rates [16, 18].

A more fine-grained scheduling problem on the downlink involves joint power control and channel allocation which allows the BS to vary the transmit power in addition to choosing how much of the channel (or bandwidth) it can allocate to the users [9], and is defined as:

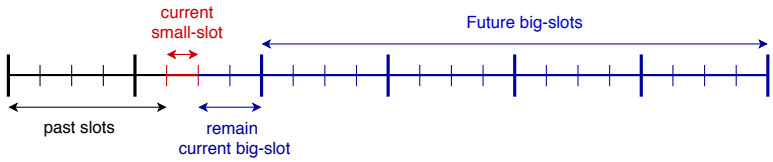


Figure 1: The different types of time slots in the STO1 algorithm.

$$\begin{cases} \text{maximize} \ \sum_{i=1}^{K} \log \left(\sum_{j=1}^{T} x_{ij} \log \left(1 + \frac{p_{ij}\gamma_{ij}}{x_{ij}} \right) \right) \\ \text{subject to} \ \sum_{i=1}^{K} x_{ij} \leq 1, \ \forall j; \quad x_{ij} \geq 0, \forall i, j \\ \qquad \frac{1}{T} \sum_{j} \sum_{i} p_{ij} \leq \bar{P}; \quad \sum_{i} p_{ij} \leq P_{max}, \forall j. \end{cases} \tag{P}$$

Here p_{ij} (resp. x_{ij}) is the power (resp. fraction of bandwidth) allocated by the BS to user i in slot j. The parameter γ_{ij} represents the channel condition of the user i in slot j and $x_{ij} \log \left(1 + \frac{p_{ij}\gamma_{ij}}{x_{ij}} \right)$ is the Shannon rate obtained by this user when it is allocated power p_{ij} and fraction of bandwidth x_{ij}. The utility of the user is again the logarithm of its total throughput and the objective of the BS is to maximize the sum utility. Note that there are constraints on the power allocation which involve both the maximum power that can be expended in a time slot as well as the average power spend over the whole horizon. For this joint power and channel allocation problem, an adapted version of STO1 was proposed in [19].

Remark 1 (Joint power control and channel allocation)**.** *For brevity, we shall explain the STO1 algorithm and its associated machine learning solution only for the channel allocation problem* (I)*. A remark shall be made wherever the treatment of this problem differs from that of the joint power control and channel allocation problem* (P)*.*

2.1 The STO1 algorithm

STO1 is a sequential algorithm which computes the allocation on time-slot j based on the current and the past data rates, and the estimated future data rates. It operates on two time-scales (shown in Figure 1) in order to reduce its complexity. Define big-slots as a certain number (order of hundreds) of time-slots (or small-slots) that reflect the duration over which the distance of each user to the BS does not change much, from that the data rate of each user in each time-slot can change a certain amount but the sum of its data rate over big-slot does not change much by law of large number.

For example, while the scheduling time-slots are 2 ms in length, one big slot can be equal to 100 time-slots, and one user can move at most by a few meters in a big slot. Big slots is defined to reduce dimension of the original optimization problem as explained below. It shall be assumed that estimations are available for users' future positions at the granularity of big-slots, and that the mean of future rates based on the future positions are estimated.

Before giving a formal definition of the STO1 algorithm, we give an intuitive explanation. In each small-slot j, STO1 does two steps. In the first step, it solves the problem (I) but with several restrictions: *(i)* the horizon is shortened to J big-slots; *(ii)* the future allocations are computed only on the aggregated level of big-slots; and *(iii)* the integer constraints on α_{ij} are relaxed. In the second step, the fractional allocation for the current small-slot is projected onto the set of the feasible integral allocations. The first step reduces the number of variables and hence the dimensionality of the problem as the future allocations are computed only for big-slots. Note that the second step is optional and is relevant only to problems with integral constraints.

A more formal definition is as follows. Let δ be the size of the small-slot, and let Δ be the size of the big-slot in absolute time units. Let $m = \Delta/\delta$ be the number of small slots in a big-slot (see Figure 1). Denote by $\bar{r}_{ij}$ the mean rate in slot j for user i. At each small-slot t, with a slight abuse of notation, we shall denote by $\bar{\rho}_{i,0} = \sum_{j=t+1}^{(m-(t \mod m))+t} \bar{r}_{ij}$ the total rate for user i in the remaining channel allocation slots of the current big-slot $\tau = 0$, where $t \mod m$ denotes the remainder when dividing t by m. We also define $\bar{\alpha}_{i,0}$ as the corresponding allocation for the current big-slot $\tau = 0$.

Denote by $\bar{\rho}_{i\tau} = \sum_{j=(\tau-1)m+A+1}^{\tau m+A} \bar{r}_{ij}$ where $A = \left(\lfloor \frac{t}{m} \rfloor + 1\right) m$, is the total average data rate that user i will get in the future big-slot τ ($\tau = 1, 2, ..., J-1$), where big slot τ starts after the current big-slot, and J is the short time horizon in term of big slots over which we can estimate the mean future rate. We also define $\bar{\alpha}_{i\tau}$ as the corresponding allocation for user i in future big slot τ. These allocations $\bar{\alpha}_{i\tau}$ can be interpreted as the fraction of small slots that user i will be allocated in the big-slot τ.

Note that this definition is slightly different from the definition in [18]. The differences are as follows: in [18], there is no current big slot and the future big-slot starts just after the current small slots, but it does not change much. The above definition of two time slot types corresponds in fact to the ones introduced in [19].

Denote by $a_i(t) = \sum_{j=1}^{t} \alpha_{ij} r_{ij}$ the total throughput allocated to user i up to time slot t, and let $K(t)$ be the number of users inside the coverage range of the BS at time t.

The algorithm STO1 contains two steps which are as follows:

- **Step 1**– solve the following optimization problem over a short-term

horizon of J big-slots:

$$\begin{cases} \text{maximize} & \displaystyle\sum_{i=1}^{K(t)} \log\left(a_i(t-1) + \alpha_{it} r_{it} + \sum_{\tau=1}^{J} \bar{\alpha}_{i\tau}\bar{\rho}_{i\tau}\right) \\ \text{subject to} & \\ & \sum_{i=1}^{K(t)} \alpha_{it} = 1, \\ & \sum_{i=1}^{K(t)} \bar{\alpha}_{i\tau} = 1, \quad \tau = 0, \ldots, J-1, \\ & \alpha_{it}, \bar{\alpha}_{i\tau} \in [0,1],, \quad \tau = 0, \ldots, J-1,\ i = 1, \ldots, K(t). \end{cases} \tag{STO1-Opt}$$

The decision variables in Problem (STO1-Opt) are the channel allocations in the current small slot, α_{it}, and the channel allocations in the current and future big-slots, $\bar{\alpha}_{i\tau}$. Since the future allocations are only computed on the time-scale of big-slots, there is a reduction by factor m in the number of variables in (STO1-Opt).

- **Step 2** – obtain a feasible integral allocation from the fractional one. For example, set $\alpha_{i^*j} = 1$ with $i^* = \text{argmax}_i\, \alpha_{ij}$ and set it to 0 for the other users.

Remark 2. (STO1-Opt) *is solved thanks to the python package CVXPY [4] and the solver MOSEK [1]. In [18], this optimization problem was solved using a projected gradient algorithm, since it allows to iteratively solve a convex optimization problem when the feasible set is a simplex or a Cartesian product of simplices, no matter how complex the objective function is as long as it is smooth and convex. The advantage of CVXPY [4] is that it can be used to generalize this idea to other convex optimization problems, with more complex constraints such as power control constraints or others QoS constraints (e.g. delay constraints).*

2.2 Learning STO1 with DFNN

As presented in [18] and [19], STO1 and STO2 are better than the other existing algorithms. However they have to solve an optimization problem with a large number of variables and constraints frequently (every small slot for STO1 or every big slot for STO2), so even if their performance is good, their computations are heavy. When the system is large and requires many more QoS contraints, they may not be able to run in real time. In addition, even when they are able to run in real time, it is good to reduce the computation time without reducing too much the quality of the allocation.

Therefore in this paper, our objective is to learn the STO1 algorithm using Deep Feedforward Neural Networks to obtain a new algorithm that behaves like STO1 but with a significantly reduced computation time. In other words, we want to learn the input-output relationship of STO1, by approximating the input-output mapping of STO1 with a DFNN (see [5] for

background material on supervised learning with DFNN). After getting the approximation function (that is, the DFNN), the output can be computed by feeding the DFNN with the input value, instead of solving an optimization problem. This simpler method is expected to work faster than the original algorithm.

The main idea is to train a DFNN to predict an approximate solution to (STO1-Opt) instead of using a specialized convex optimization package. Obviously, the same idea could be used for other problems in order to obtain an approximate method which performs almost as well as the original algorithm but requires much less computing time. In short, the approach is as follows:

1. Design a well-performing algorithm based on the best available information;

2. Learn it by an approximate algorithm which behaves closely to the original one and has less computation.

The basic idea can be summarized as follows. Supervised learning is a learning task that amounts to learning an input-output relationship from a bunch of examples of input-output pairs which is called training data. The relationship can be derived from those examples by analyzing the training data and an inferred function will then be produced. A learning algorithm is considered as good if it can generalize the input-output relationship, i.e, it is able to determine outputs of unseen inputs with small error. So the objective here is to find the function representing as well as possible the input-output relationship.

Mathematically, let F be the true function that represents the relation between input and output. However, if the true function is unknown, our task is to find an approximate function $\hat{F}$ that is inferred from examples taken from training data $(x_n, y_n)_{n=1}^N$. The STO1 algorithm can be seen as a function F that maps an input $x_n \in \mathcal{X}$ (a problem instance) to an output $y_n \in \mathcal{Y}$ (a channel allocation), where $\mathcal{X}$ and $\mathcal{Y}$ denote the input and output spaces of STO1, respectively. Unfortunately, STO1 is too complicated to get the exact formula of F. Therefore here we want to approximate it by another function $\hat{F} : \mathcal{X} \to \mathcal{Y}$ which is in the form of a DFNN (an example of DFNN is illustrated in Figure. 2). Our objective is thus to find $\hat{F}$ such that it minimizes the empirical risk

$$R_{\text{erm}}(F) = \frac{1}{N} \sum_{n=1}^{N} l(y_n, F(x_n)),$$

where $l(\cdot)$ is a loss function which measures how far y_n is from $F(x_n)$. Note that $l(\cdot)$ is not necessarily a mathematical distance but it is similar to a distance in the sense that it is small if the difference between y_n and $f(x_n)$ is small.

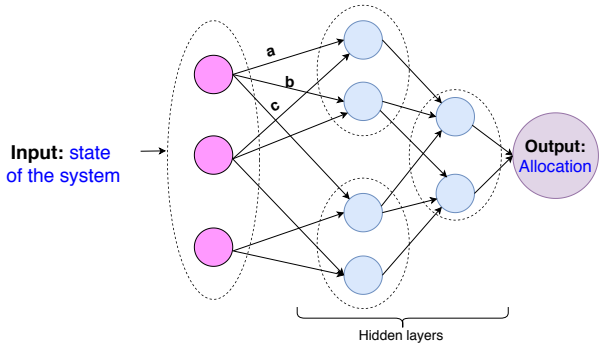


Figure 2: An example of Deep Feed-forward Neural Network.

A DFNN is composed of many linear functions (sum of matrix multiplications and bias vectors) and non-linear functions (relu, sigmoid, softmax, etc.), and inside linear functions there are many parameters. So finding a good DFNN function means finding a good architecture, that is: the way the linear and non-linear functions are combined, the linear and non-linear functions in each layer, the size (number of units in each hidden layer) and then their parameters. Finding a good architecture is in general not an easy task [22]. In this paper, we shall empirically compare some architectures through experiments presented in Section 4.2. After fixing the architecture, we have to find appropriate parameters by minimizing the empirical risk defined above.

3 System Setup for learning

Recall that, in STO1, we have two types of time scales: one is the big slot Δ and the other one is the small slot δ. In STO1, we solve problem (STO1-Opt) with variables of size $(1+J)*K$ every small slot, where J is the time horizon in terms of big slots and K is the number of users in the system. The size of the allocation vector for each user is equal to $1+J$ since it contains the allocation for the current small slot, α_{it}, and the average allocation $\bar{\alpha}_{i\tau}$ for the subsequent J big slots (including the current big slot). The input of STO1 is a data rate vector of size $(1+J)*K$ and the total allocated throughput for the K users. The output of STO1 is the current allocation vector $(\alpha_{it})_{i=1,\dots,K}$ which is of size K, since we shall only use current allocation for making decision.

As it is defined at the moment, STO1 is not well suited to be modelled as a learning problem for the two reasons stated below.

Firstly, since K can vary over time, the dimension of the input vector will also vary. To circumvent this problem and to properly define STO1 as

a function, we have to fix the size of the state. To do that, we extend the real state of the system by adding some pseudo users. Let us assume that there are at most K_M users inside the system. We will then add $K_M - K$ pseudo users, where K is the number of real users in the system at time t. We will actually learn an extended version of STO1 which is STO1 when we restrict it to K users. There are many ways to extend STO1, but here we try to define an extended version that preserves as much as possible the continuity of STO1. When we mention "learning STO1", it means "learning the extended function" of STO1.

Secondly, the output of STO1 as defined above is the solution of an optimization problem. So in fact STO1 is a set-valued mapping since the solution need not be unique. But by using the CVXPY package to solve the convex optimization problem (STO1-Opt), we agree with the way it determines one of the solutions. This makes STO1 becomes a function (instead of set-valued mapping).

Remark 3 (Joint power control and channel allocation)**.** *For the joint power control and channel allocation problem, the state needs to be augmented by the remaining total power. The output of the DFNN will now give the transmit power to each user as well as the fraction of the channel it gets allocated.*

3.1 State

We define a state as a matrix of size $(2+J)\times K_M$, where K_M is the maximum number of users in the system. There are thus $2+J$ rows, and each row has K_M elements. The interpretation is as follows:

- The first row gives the current rates of the K users. We fill in the K positions on the left hand side with these current rates, and the remaining $K_M - K$ positions are filled with -1.
- The next J rows (from $2, .., J+1$) give the average rates of the users in the next J big slots. For pseudo users (the $K_M - K$ columns on the right hand side), we use the value $(-1)\cdot(\Delta/\delta - (t \mod \Delta/\delta))$ for the current big slot and the value $(-1)\cdot\Delta/\delta$ for the other big slots.
- The last row gives the total allocated throughput of the K users. For pseudo users, we use a large enough value which is significantly greater than the total allocated throughput of real users.

By observing how STO1 works, we remark, as expected, that STO1 gives priority to users with a low allocated throughput and a high current rate. Therefore, the way we define the state (that is, by using negative values for the current and future rates of pseudo users, and extremely large values for their allocated throughput) is intended to help the model ignore quickly the pseudo users.

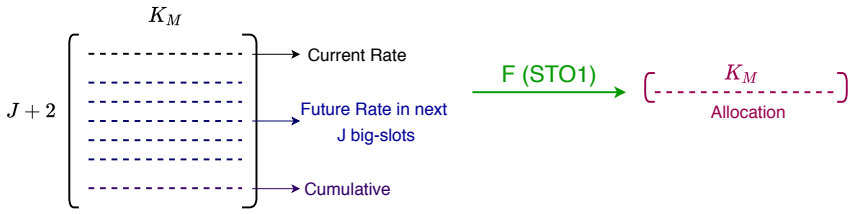


Figure 3: Input and output of the DFNN model.

Remark 4. *Remark that there are K real users in the system at present time. Therefore, the K places of the real users in the first row which give the current rates of those users have to be strictly positive. The future rates (from the second row to the $(J+1)$-th row) can be zero.*

3.2 Target

We remind the reader that we want to learn only the current allocation, not the future allocation. Therefore, the target will be a vector of size K_M, where the first K positions represent the fractional allocation α_{it} of the K users as computed by STO1, and the last positions are filled with zero. Since in the optimization problem, the sum of allocation should be equal to 1, when there is no user in the system (all positions correspond to pseudo users), the allocation vector will be set to $(1/K_M, ..., 1/K_M)$ by convention.

Figure 3 illustrates the input and output of the DFNN model as described above.

3.3 Loss, DFNN architecture, initial parameters and optimizer

We will try several different loss functions and architectures and compare them in the numerical section. The initial parameters (weights) of the DFNN will be chosen as proposed in [14], which allows the initial parameters to be not too big and not too small. The optimizer is Adam, which was first introduced in [11] and is a stochastic first-order gradient-based algorithm. The convergence of Adam is proven in [20].

4 Numerical Comparisons

In this section, we do simulations to evaluate the influence of many factors on the behavior of the DFNN model (loss functions, architecture of the DFNN). We use the keras library [2] to implement our code.

There are actually a lot of factors that can have an impact on the behavior of the learning procedure such as the initial learning rate, the learning rate decay, the optimizer, the initial weight, the number of parameters, the

activation functions in layers... Here we are not able to justify all our choices, but we focus on the factors which have the most significant impact on the learning algorithm in our opinion. The initial learning rate is chosen equal to 0.0015 and after each epoch, this learning rate decays by a factor 0.998.

4.1 An Unified Data Generator for Comparison

To support the comparisons in this section, the data (both for training and validation) is generated as follows. The number of users is generated randomly from 0 to $K_M = 10$. The sojourn time of each user is generated in $(0, 400)$ seconds. This value could of course be increased, but here in order to reduce the learning time and be able to make many comparisons, we consider only small scenarios. At each learning epoch, the model will go through 1600 samples (that is, input-output pairs). The transmission rate in each small slot is generated randomly between 0 and $5 * \delta/\Delta$. The rate we use for evaluating in SUMO scenarios are given by

$$r(x) = \eta \left(1 + \kappa\, e^{-d(x,BS)/\sigma}\right), \tag{1}$$

where $d(x, BS)$ is the distance from position x to the BS, and η represents the noise level. For the SUMO scenarios in Section 4.4, we use $\kappa = 3$, $\sigma = 100$ and $\eta \sim \text{Uniform}(0.7, 1.3)$. The others parameters are equal to $J = 10, \Delta = 1$ s, $\delta = 2$ ms.

4.2 Comparison of different DFNN architectures

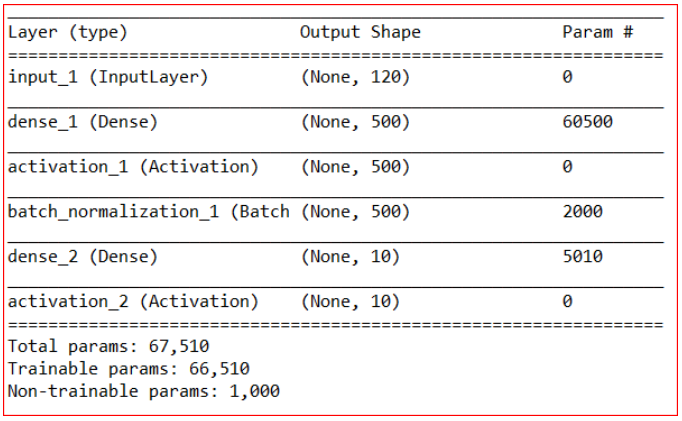
In this part, we will consider 4 different architectures of the DFNN model and compare their performances. For the 4 models, the activation function used in hidden layers is the relu function, whereas the output layer uses the softmax function since we want the sum of the allocations to be equal to 1. In this comparison, we use the same loss function for all models, the huber loss [26].

4.2.1 Model 1

The first model used in this section contains 2 layers which are 1 hidden layer and 1 output layer. The hidden layer contains 500 units, and in total the model has $67,510$ parameters. The architecture of this model is illustrated in Figure 4.

4.2.2 Model 2

As the first model, the second model contains 2 layers: 1 hidden layer and 1 output layer. However, the hidden layer contains 1000 units, and in total the model has $135,010$ parameters. We take the same number of layers as in



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 120)	0
dense_1 (Dense)	(None, 500)	60500
activation_1 (Activation)	(None, 500)	0
batch_normalization_1 (Batch	(None, 500)	2000
dense_2 (Dense)	(None, 10)	5010
activation_2 (Activation)	(None, 10)	0

Total params: 67,510
Trainable params: 66,510
Non-trainable params: 1,000

Figure 4: Model 1 architecture.

model 1 (but with more units in hidden layers) in order to compare whether it is better to have more parameters.

4.2.3 Model 3

As the two previous models, the third model contains 2 layers (1 hidden layer and 1 output layer). The hidden layer contains 100 units, and we have $13,510$ parameters in total. We take the same number of layers as in model 1 (but fewer units in the hidden layer) to compare whether it is better to have fewer parameters.

4.2.4 Model 4

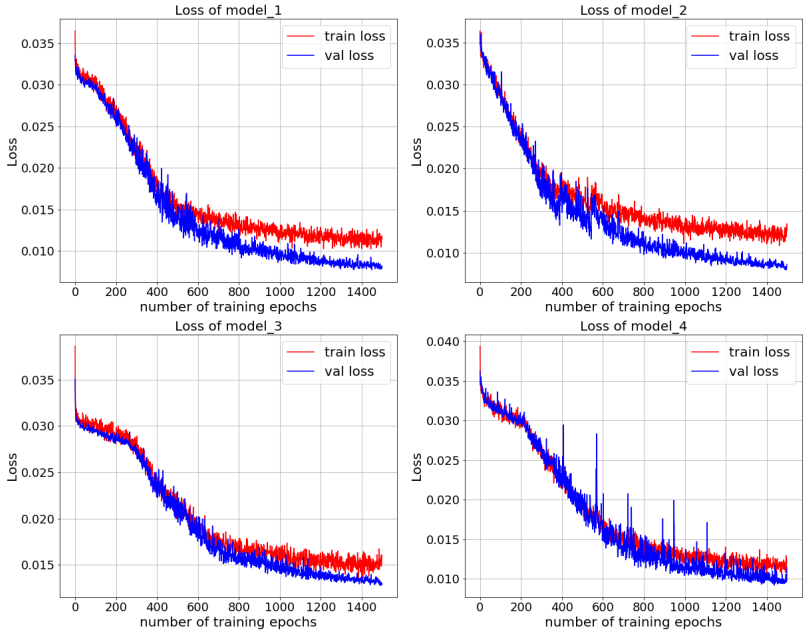
The last model contains 10 layers which are 9 hidden layers and 1 output layer. Each hidden layer contains 82 units, and in total the model contains $67,496$ parameters. We take a model that has almost the same number of parameters as Model 1, to compare whether it is better to have more layers or fewer layers.

Remark 5 (Joint power control and channel allocation)**.** *For the joint power control and channel allocation problem, we still compare the four above models except that the output layer of each model will be modified since it includes not only the channel allocation but also the power.*

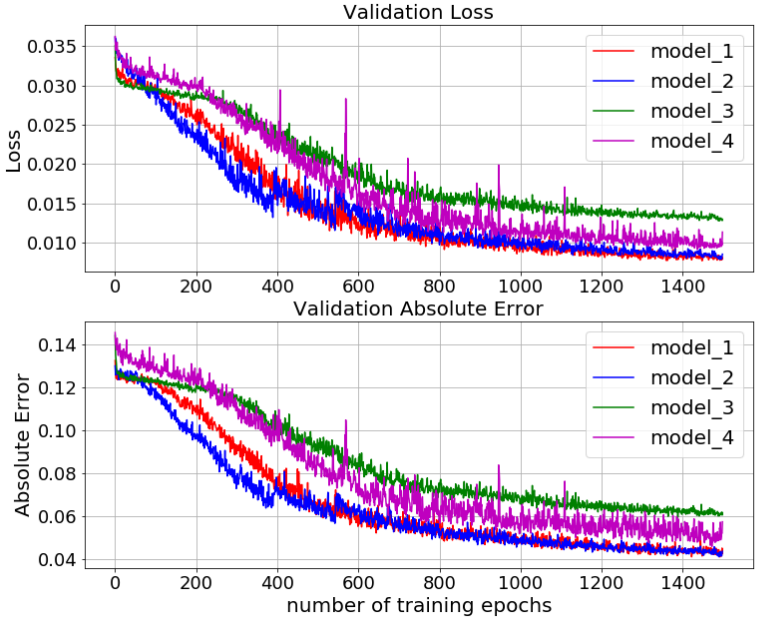
Figure 5a illustrates the loss of the 4 models on training and validation data. Figure 5b plots loss and absolute error of the 4 models on the same axis on validation set. The same quantities but for the problem of joint power control and channel allocation are shown in 6.

From these figures, we observe that for the model without power control:

- Having almost the same number of parameters, Model 1 with fewer layers is better than Model 4.



(a) Loss on training set and validation set of each model



(b) Plot on same axis for loss and absolute error on validation set of all the four models

Figure 5: Comparison of the 4 DFNN models.

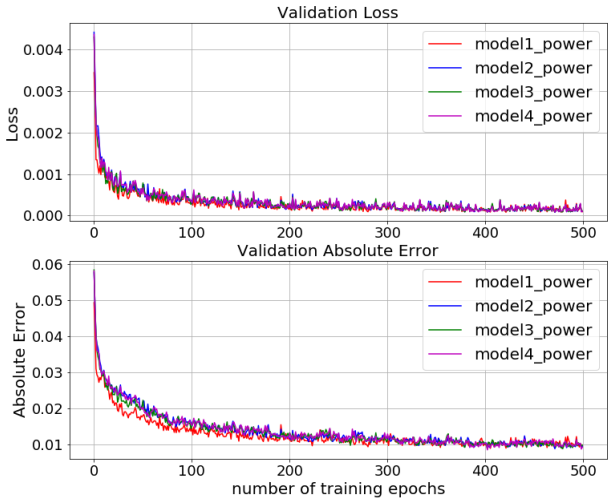


Figure 6: Comparison of the 4 DFNN models for the joint power control and channel allocation problem.

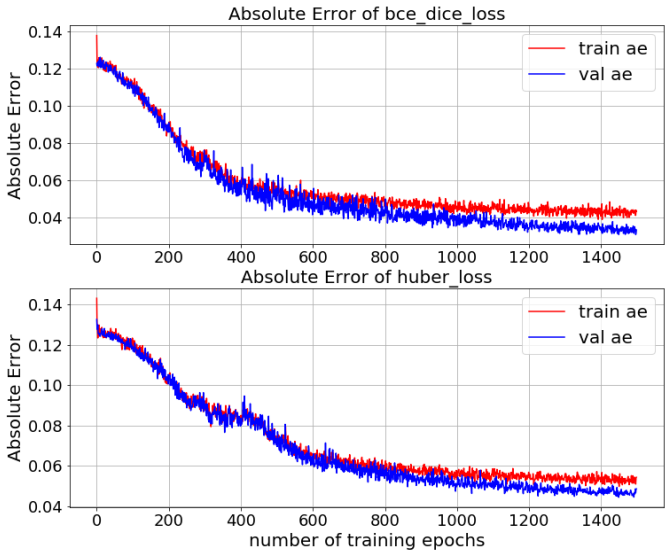
- Having the same layers, Model 1 and Model 2 with more parameters are better than Model 3.
- Model 1 and Model 2 behave similarly and have the same number of layers. However Model 1 has less parameters than Model 2 so it is less costly from a computational point of view. Therefore from now on we shall use Model 1 for other comparisons in the sequel.

For the model with power control:

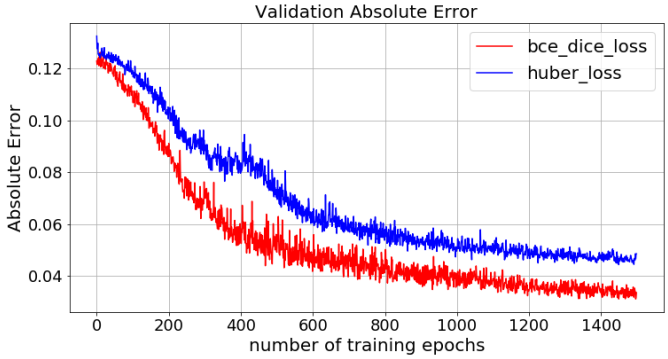
- Having almost the same number of parameters, Model 1 with few layers is slightly better than Model 4, but the difference is quite small in this case.
- Having the same layers, models 1, 2 and 3 are almost the same but Model 3 has fewer parameters.

4.3 Comparisons of different loss functions

To compare the quality of the learning model obtained using different loss functions, we use the same model, that is Model 1. Figure 7 presents the results obtained with the huber loss [26], with the sum of binary cross-entropy [24] and dice loss (which equals to 1− dice coefficient [25]). The second loss function is denoted by bce_dice loss. From the figures, the bce_dice loss function is better in this case. So for the next comparisons, we shall use Model 1 and bce_dice loss.



(a) Absolute error on training and validation set of the two losses



(b) Plot on same axis of the absolute error on validation of the two losses

Figure 7: Comparison of Loss functions.

Figure 8: The Carmes borough in Toulouse, with one BS (Free Mobile type LTE1800). The actual size is $200m \times 400m$.

4.4 Performance Evaluation on SUMO scenarios

In this section, we shall use a mobility simulation software for comparisons of the algorithms, that is Simulation of Urban Mobility application (SUMO) [15]. SUMO is an open source software designed for simulating mobility of moving users (vehicles, bus, truck, bicycle, pedestrian, ...) in large road traffic networks. It allows to import maps of different cities and simulate realistic mobility traces. This application is used to simulate the complex moving dynamic systems in several specific regions of Toulouse city to compare our heuristics against existing algorithms in realistic scenarios. The performance evaluation of the heuristic is done in two steps: firstly, SUMO is used for generating the mobility traces of vehicles; finally, these traces are then fed to a Python script which implements the algorithms and computes the value of the objective function of those algorithms.

We shall compare the learning-based allocation scheme with STO1 and other existing algorithms on two different scenarios created with SUMO. The first scenario contains 244 users and lasts 61.7 minutes. The map of this scenario is shown in Figure 8. The results obtained with the different learning schemes on this scenario are shown in Figure 10. The second scenario contains 214 users and lasts 62.4 minutes. The map of this scenario is shown in Figure 9. The data for BS location can be found on the website[3] of the French Frequency Agency (ANFR), which manages all radio frequencies in France. The results obtained with the different learning schemes on this scenario are shown in Figure 11. We also simulate two existing algorithms, $(PS)^2S$ [16] and PF [10] (which are also used in [18] for comparisons) in order to show that the approximation algorithm performs better than the existing algorithms. As mentioned above, for the learning algorithm, we use Model 1 and bce_dice loss.

When the number of learning epochs is large enough, the learning-based scheme performs well compared to STO1 and other algorithms.

[3]https://data.anfr.fr/anfr/portail

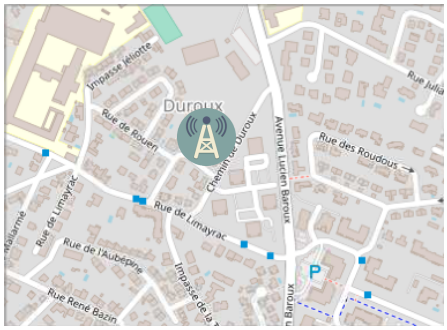


Figure 9: Duroux, one BS type LTE1800, operator SFR. The actual size is around $350 \times 500m$.

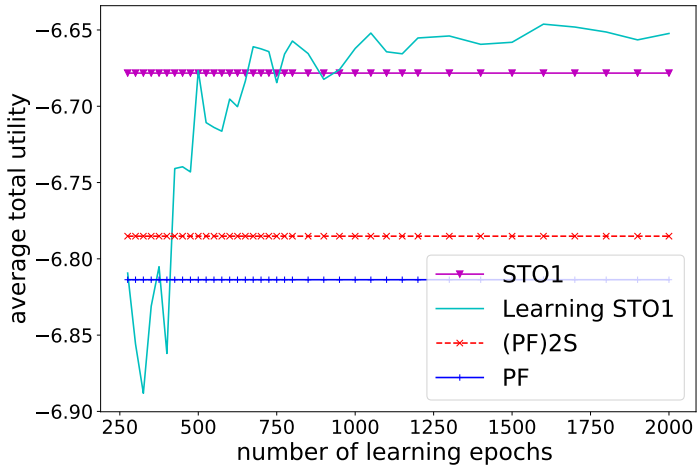


Figure 10: Comparisons of evaluated on Carmes scenario created by SUMO.

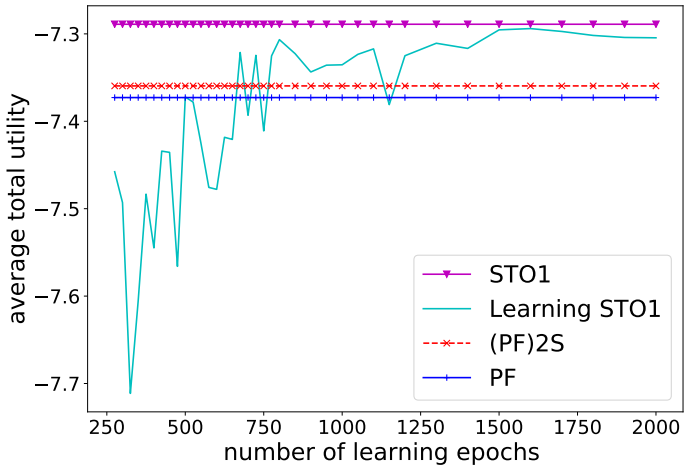


Figure 11: Comparisons of evaluated on Duroux scenario created by SUMO.

5 Computing times

The computing time of STO1 depends on the convex optimization solver used, whereas the learning algorithm has only to feed the DFNN model with the input matrix. We consider the same setting as in Section 4.2, that is $K = 10$ (there are 10 users in the system) and the short term horizon is $J = 10$ seconds. For these values, the average computing time of Mosek is around 43.7 ms, whereas the prediction with the DFNN model (Model 1, which contains 67,510 parameters) takes only 0.65 ms on average. When adding power control, Mosek solves the optimization problem in around 113.4 ms, while the prediction of the DFNN model (almost the same with Model 1 but the output layer contains power vector in addition, which contains 73,020 parameters) takes 0.68 ms on average. These computing times are averaged over 10000 samples, all are measured on a machine using GPU (graphics processing unit) which allows computing many calculations in parallel.

From the above measurements, we can conclude that the computing time with a solver can vary widely with the number of QoS constraints for the same network (number of users, time horizon). In contrast, the prediction time of the learning-based algorithm with DFNN is almost insensitive to such changes.

6 Summary and Discussion

We have proposed to use DFNN for learning the channel allocation obtained with one of the heuristics (STO1) introduced in [18] and [19]. Numerical

results on SUMO scenarios show that the learning-based method yields approximate yet satisfactory channel allocations with much less computation time as long as there are enough learning epochs. The state of the DFNN is defined in such a way that the model is not restricted to a particular scenario, that is, it can learn the channel allocation for a general network.

There are several directions of research that can be investigated to improve the learning algorithm, such as a better generator of data, a better loss function, a better architecture of the DFNN model, and other things such as the optimizer, the learning rate, etc.

7 Acknowledgements

This work was partially funded by a contract with Continental Digital Services France.

References

[1] M. ApS. *MOSEK Optimizer API for Python manual. Version 9.2.21*, 2019.

[2] F. Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[3] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[4] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[5] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[6] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 399–406, Madison, WI, USA, 2010. Omnipress.

[7] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[8] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989.

[9] J. Huang, V. Subramanian, R. Agrawal, and R. Berry. Downlink scheduling and resource allocation for ofdm systems. *IEEE Transactions on Wireless Communications*, 8:288–296, 01 2009.

[10] F. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8(1):33–37, 1997.

[11] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[12] A. Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Dokl. Akad. Nauk SSSR*, 114(5):953–956, 1957.

[13] H. J. Kushner and P. A. Whiting. Convergence of proportional-fair sharing algorithms under general conditions. *IEEE Transactions on Wireless Communications*, 3(4):1250–1259, July 2004.

[14] Y. Lecun, L. Bottou, G. Orr, and K.-R. Müller. Efficient backprop. 08 2000.

[15] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[16] R. Margolies, A. Sridharan, V. Aggarwal, R. Jana, N. K. Shankaranarayanan, V. A. Vaishampayan, and G. Zussman. Exploiting mobility in proportional fair cellular scheduling: Measurements and algorithms. *IEEE/ACM Trans. Netw.*, 24(1):355–367, Feb. 2016.

[17] H. Montanelli and H. Yang. Error bounds for deep relu networks using the kolmogorov–arnold superposition theorem, 2019.

[18] N. Nguyen, O. Brun, and B. Prabhu. An algorithm for improved proportional-fair utility for vehicular users. *The 25th International Conference on Analytical and Stochastic Modelling Techniques and Applications ASMTA-2019*, May 2019.

[19] N. Nguyen, O. Brun, and B. Prabhu. Joint downlink power control and channel allocation based on a partial view of future channel conditions. *The 15th Workshop on Resource Allocation, Cooperation and Competition in Wireless Networks*, June 2020.

[20] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of adam and beyond. *CoRR*, abs/1904.09237, 2019.

[21] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos. Learning to optimize: Training deep neural networks for wireless resource management. *CoRR*, abs/1705.09412, 2017.

[22] S. Sun, W. Chen, L. Wang, X. Liu, and T.-Y. Liu. On the depth of deep neural networks: A theoretical view. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2066–2072. AAAI Press, 2016.

[23] D. Tse and P. Viswanath. *Fundamentals of Wireless Communication*. Cambridge University Press, 2005.

[24] Wikipedia. Cross entropy. 20 June 2020. `https://en.wikipedia.org/wiki/Cross_entropy`.

[25] Wikipedia. Sorensen dice coefficient. 28 July 2020. `https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient`.

[26] Wikipedia. Huber loss. 29 May 2020. `https://en.wikipedia.org/wiki/Huber_loss`.

[27] Y. Yi and M. Chiang. Stochastic network utility maximisation—a tribute to kelly's paper published in this journal a decade ago. *European Transactions on Telecommunications*, 19(4):421–442, 2008.