Diploma Thesis

Network-aware Distributed Deep Neural Networks for Slice Resource Allocation in 6G

Bc. Ondřej Šmíd

Supervisors:
prof. Thrasyvoulos Spyropoulos, Ph.D.
doc. Ing. Zdeněk Bečvář, Ph.D.

Fakulty of Electrical Engineering

Department of Telecommunication Engineering

January 3, 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šmíd** Jméno: **Ondřej** Osobní číslo: **469848**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra telekomunikační techniky**

Studijní program: **Elektronika a komunikace**

Specializace: **Komunikační sítě a internet**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Alokace zdrojů pro slicing v 6G sítích pomocí distribuované hluboké neuronové sítě

Název diplomové práce anglicky:

Network-aware Distributed Deep Neural Networks for Slice Resource Allocation in 6G

Pokyny pro vypracování:

Architektury DNN (Deep Neural Network) lze použít k předpovídání provozu základnových stanic a podle toho alokovat zdroje, aby se minimalizovaly náklady na nedostatečné a nadměrné zřizování. Přesto provozování velkých DNN v cloudu edge představuje pro sítě 5G+/6G značné výzvy související s latencí, velkým množstvím dat shromážděných na základnových stanicích nebo složitostí a velikostí DNN. Cílem této práce je prozkoumat vhodnost distribuovaného DNN (DDNN) prováděného na okraji mobilní sítě. DDNN by mělo být zvažováno společně s definicí skutečného řezu s více VNF vyžadujícími zajištění zdrojů na případně různých hraničních místech. Dále by měl být nastíněn a vyhodnocen automatizovaný mechanismus ladící offline i online parametry DDNN za letu podle scénáře.

Seznam doporučené literatury:

[1] D. Bega et al., “DeepCog: Cognitive Network Management in Sliced 5G Networks with Deep Learning,” in Proc. of IEEE Infocom 2019.
[2] S. Teerapittayanon, B. McDanel, H.T. Kung, “Distributed Deep Neural Networks over the Cloud, the Edge and End Devices,” https://arxiv.org/abs/1709.01921

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Zdeněk Bečvář, Ph.D., katedra telekomunikační techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **15.09.2021** Termín odevzdání diplomové práce: **04.01.2022**

Platnost zadání diplomové práce: **19.02.2023**

doc. Ing. Zdeněk Bečvář, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

I declare I have accomplished my diploma thesis by myself and I have named all the sources used in accordance with the Guideline on ethical preparation of university theses.

In Prague, ...

Signature:

Acknowledgment

First and foremost, I have to thank my supervisors, prof. Thrasyvoulos Spyropoulos, Ph.D. for patience during long and often meetings and doc. Ing. Zdeněk Bečvář, Ph.D. for his guidance during the writing. Due to their support telecommunications has become my passion. I would like to thank my colleagues from EURECOM and CTU for their support during my studies. And the most importantly I have to thank my family and especially my parents who have been supportive during all my way.

Anotace

Network slicing se ukazuje jako klíčový koncept v sítích 5G a dalších generacích. V konceptu network slicingu by operátoři sítí měli spravovat fyzické zdroje mezi slicy. Tyto fyzické zdroje, používané a pozorované na různých místech, jsou ideálně alokovány do každého slicu sítě na základě předpovědi spotřeby zdrojů. Zdroje z více lokalit jsou obvykle agregovány přes Edge v Cloudu, kde lze alokaci vytvořit, komunikace Edge-Cloud zvyšuje komunikační režii. Tato práce navrhuje formu distribuce hlubokých neuronových sítí (DNN), která snižuje míru komunikační režie v síti způsobenou přenosem informací o zdrojích Edge-Cloud při zachování schopností předpovědi centralizované DNN. Práce také navrhuje novou cílovou funkci správy sliců umožňující (D)DNN vytvářet alokaci pro více sliců najednou. Distribuce DNN vyžaduje kontrolu výkonu, pro které jsou navržena tři možná řešení. Všechna řešení jsou porovnána pro natrénované DDNN a použita pro porovnání mezi výkonem a komunikační režií. Nabízená řešení umožňují dosáhnout pouze zlomku potřebné komunikace v centralizovaném DNN při dosažení mírného poklesu výkonu nebo v některých případech i zvýšení výkonu. V některých případech jsou uvedny funkce, které ukazují skutečné snížení režie a zvýšení výkonu.

Klíčová slova: Alokace zdrojů, Network slicing, Strojové učení, Distribuované hluboké neuronové sítě

Annotation

Network slicing emerges as a key concept in 5G and beyond networks. In the network slicing concept, the networks operators should manage physical resources between slices. These physical resources, used and observed in different locations, are ideally allocated to each slice based on a resource consumption forecast. Resources from multiple locations are usually aggregated through the Edge in Cloud, where the allocation can be created, Edge-Cloud communication increases communication overhead. This work proposes a form of deep neural networks (DNN) distribution, which alleviates communication overhead in the network caused by Edge-Cloud resource information transfer while maintain forecasting capabilities of centralized DNN. The work also proposes a novel slice management objective function enabling (D)DNN create allocation for multiple slices at once. DNN distribution requires performance moderation for which three possible solutions are offered. All solutions are compared for trained DDNN and used for description of trade off between performance and communication overhead. The offered solutions enable to achieve only a fraction of communication needed in centralized DNN while reach slight performance decline or in some cases even performance gain. Trade off functions are offered in some cases to show real overhead reduction and performance gain.

Keywords: Resource Allocation, Network Slicing, Machine Learning, Distributed Deep Neural Networks

Table of Contents

List of Figures

Abbreviations

3D-CNN 3D version of CNN

BS Base Station

C-RAN Cloud Radio Access Network

CNN Convolutional Neural Network

DDNN Distributed DNN

DNN Deep Neural Network

DRL Deep Reinforcement Learning

DTW Dynamic Time Warping

FC Fully Connected Layer

LSTM Long Short-Term Memory version of recurrent neural network

MDS Multi Dimensional Scaling

MEC Multi-Access Edge Computing

MSBD Mean Shape-Based Distance

RAN Radio Access Network

ReLU Rectified Linear Unit

SBD Shape-Based Distance

SLA Service Level Agreement

UE User Equipment

VNF Virtual Network Function

1 Introduction

Mobile networks are influenced by network slicing, which promises high level of customization for industry or other fields using 5G and beyond. This level of customization starts to gain significance and importance which starts to dominate definitions of new techniques and approaches in mobile networks.

Network slice customization targets to create reliable networks under certain constrains with usually one or a few key aspects in mind. An example of the key aspects is IoT communication served by a network slice stressing reliability and latency over network throughput, which can be seen as resources in network slice or the whole network. Network operators usually operates with several resources which have to be allocated ahead of their use and operators are always seeking better ways of resource forecasting.

In general, resources are usually rare or time restricted or simply there is a will to utilize them efficiently without wasting. Either of reasons justify an importance of resource management which determines a policy how a certain resource should be used in a way that maximizes financial profit, service reliability in mobile networks, or another predetermined goal. This work takes a purpose to use a single resource (any resource in general), create and reliable allocation within some network divided by network slices. A single representative resource is used in this work and it is the network traffic load. Network traffic is observed and provision on multiple locations. A better description can be offered as to suggest a system capable to create resource allocation ahead of the use for one and/or multiple network slices.

In the nowadays communications networks, beyond 5G, network slicing is considered as a key aspect enabling to create virtually independent networks. Each network slice is assigned with some resources which its users will consume and since it is only a virtual concept it has to be considered and managed with the other slices as a part of the bigger, physical network.

To predict the user consumption of some resources in the network is still a problem. Especially in mobile networks where users have ability to change their location and thus a source of the resource. To predict which source should have resource available and how big amount of a resource should be available at on source in the whole network is a challenge. To preallocate a resource in combination with a network slicing where each network slice has its own independence is even worse. To create a reliable resource allocation is a goal of this work. Due to the problem complexity and inability of proper and full mathematical description the work uses machine learning techniques for the resource allocation.

Previous works [1] [2] [3] [4] and especially [5] has already offered solutions of a problem with resource allocation for a network where a slice has a single source which users can utilize as well as prediction of a single slice. This can be for example a single virtual network function (VNF) or a single base station (BS). But they are not offering a solution predicting resources of multiple slices at once.

This work introduces multiple network slices in the resource allocation problem, thus one network slice has multiple sources requiring resource allocation. This can be described as a network slice defined on top of multiple VNFs.

Prediction using machine learning is still computationally demanding because it needs cloud computational power to be effective in the communications networks. However, sending all necessary information for prediction is costly and creates communication overhead which can not be simply mitigated. Further, this work shows an

effective way of machine learning model distribution over multiple parts of network (Edge and Cloud) which will alleviate constraints added by the prediction model. Solution for spreading machine learning model over multiple nodes is described in [6].

The work is divided in several sections, where section 2 describes other solutions offered in resource management field, which techniques are used for solving this problem. It also offers some insight on the possible ways of distributions in machine learning. Further section 3 focuses on description of centralized DNN for effective resource allocation with data preprocessing, introduction into DNN distribution and its training and run time usage with results comparison for both approaches. Its purpose is to describe centralized DNN similar to the one used in [5] and to show that the DNN model remains its forecasting capability even with distribution, while also describes all its details. This solution offers a (D)DNN architecture capable to allocate resources for one slice. After introducing one slice solution further section 4 is dedicated to promote changes in objective function enabling the model create allocation for multiple slices at once. It presents important definitions for objective function used in model training with respect of slice management, namely recognizes difference between objective function and penalty policy which combination form a loss function used in (D)DNN training to ensure specific goals in DNN use. The next, section 5, speaks closely about a key mechanism in the distributed model. This key part decide on the allocation performance and the network communication overhead reduction trade off. Its higher performance benefits in the overall DDNN performance. It briefly introduces theory of the possible solutions and offers three concrete solutions. At the end of this section, detailed comparison is offered to show the performance of the whole model influenced by the combination of different decision process and objective function with certain penalty policy. The last two parts of the work offer possible future enhancements of the DDNN or the work and directions of research to exploit findings of this work in use for reinforcement learning **??** and the conclusion, where benefits of the work and the topic are balanced and summed up section 6. The conclusion offers short list of the work findings necessary for creating distributed DNN capable of producing automated accurate resource allocation for multiple slices at once.

2 Related Work

Resource allocation for network slicing is a discussed topic since the network slicing have been offered [7]. Resource allocation management can be done by two approaches, where one is reservation based and the other is shared based [8]. Authors in this paper offers comparison of these two approaches, where reservation based approaches forces tenants to predetermine their needs while operators have to block their resources strictly for use of only one tenant. [8] offers comparison with dynamic approaches where tenants shares operator resources for several key conditions: customizability, complexity, efficiency, isolation, privacy and cost predictability. It can be said that scalability is the main problem in reserved based cases, where the customizability is the biggest problem as well as cost predictability, which depends on billing scheme of the SLA. Intuitively cost predictability is not an issue in this case, but in comparison with shared based approaches where tenants pay for their actual use by the predetermined price reserved based approach pricing is more complex. Scalability is an issue for shared based approaches only by limitation of overall operator resources capability, but it lacks in isolation which in case of reserved based approaches is implicit.

Further, authors of [8] identify ways of realization for both classes of approaches. Both classes have to allocate resources ahead of it use where machine learning techniques are identified as forecast capable group of algorithm performing demand prediction based on the resource demand history.

Machine learning techniques are widely utilize seemingly in any field nowadays. Wireless communications are not an exception in this because machine learning techniques offer range of benefits in their generalization capabilities. They can offer solution where mathematical solution is not known. Machine learning has its downside in computational demand but mainly in its time consumption it needs to be allocated for development and training of machine learning models. Even if it is burdened by this drawback, it is still widely used because of the unique and important results it is capable to provide. This section presents a brief presentation of related works and at the end presents a similar small overview for DNN distribution.

Resource allocation management in wireless communications is a subject of research with machine learning crossover as suggests [1]. The survey offers a machine learning techniques and theirs respective use in certain domains of wireless communications. From the survey it can be seen several techniques are used for resource management. The survey presents that the main focus in the papers has been given to some techniques, namely autoencoders, deep learning and lately deep reinforcement learning.

Deep learning uses either LTSM or CNN and for example [2] utilized LSTM to extract spatiotemporal features of traffic load. LSMTs are widely used for forecasting and in related works are also used in combination with CNN or 3D-CNN [4].

Use of CNN is justified in neural network called *DeepCog* [5] which should extract information about correlation features from provision of slices in the neural network input. The same group of authors then combined benefits of provision correlation extracted by 3D-CNN in LSTM utilizing neural network called *Aztek* [3].

However, deep learning are used for research, in practice it has a significant sat back in its static nature where the training is needed for each specific task. More so if CNN is used it depends on static patterns in data which the model learnt during the training and if data patterns fade away or are replaced by new ones in time the

model needs to be retrained or its accuracy and performance drops. Usually works count on long term changes in data fed to the models but one group of machine learning based works describes technique capable of seeing and adapt to changes in the data set. This is reinforcement learning and some work has been done to harvest the potential of combining deep and reinforcement learning in what is called deep reinforcement learning (DRL) [9], this work focuses on slice placement optimization over the physical network.

The only work (by the best knowledge at the time of writing) which is trying to solve resource allocation problem using DRL is [10]. Authors are actually using D-DRL - distributed DRL which makes this work also the only one trying to distribute machine learning model over several nodes by using distributed agent. Even though the method of machine learning technique and distribution method is different the work represents an interesting approach to the resource allocation management problem.

It has to be admitted that DRL using structure described in this work would be very interesting solution adding another angle of distribution while enjoying the benefits of reinforcement learning like data adaptability. But it is not an interest of this work and first it has to be shown that deep learning variant is capable to offer reliable solution.

Even though this is not the first work offering the distribution of resource allocation management machine learning model it is the first work offering distribution presented in [6] [11]. Authors of these papers distributed machine learning model, a classifier, consisted of several layers by isolating parts of layers and details are discussed in subsection 3.3.

3 System Model

This section describes centralized DNN for resource allocation, how DNNs can be used for time series forecasting or resource allocation . It also presents dataset used for training and testing the DNN and a way data are processed into the DNN model input. After a description of the centralized DNN section presents distribution of DNN over multiple nodes in the network and importantly it explains how to train distributed DNN. This section conclude its description by performance comparison between centralized and distributed DNN to show benefits of DDNN use.

3.1 Resource Allocation

As it is previously stated resource allocation is a problem addressed in multiple previous works and solutions using machine learning techniques vary. The problem all works, and even this one, target to solve is to create a prediction of certain time series. Usually ML models for prediction problems use forms of LTSM or CNN or its combinations. LSTMs generalize history of the time series, on the other hand CNNs work with multiple time series at once and extract information about correlation between time series. This work utilizes 3D-CNN, which is 3D version of the CNN enabling the network extract information about a history in the input.

Lets consider a set of time series, for more concrete abstraction a set of $\mathcal{M}$ VNFs, which each of them uses some resources, for example a traffic demand. A single traffic sample at time t of a single VNF $i \in \mathcal{M}$ is denoted d_t^i, then d_t denotes a set of traffic samples at time t of all $\mathcal{M}$ VNFs. To extract information from traffic history for each VNF multiple time instances of these samples d_t has to be considered for predicting next time step sample d_{t+1}. Further described DNN considers only limited history *window* of length N and input is then a series of past samples denoted $\boldsymbol{d}_{t,N} = \{d_{t-N}, ..., d_t\}$, which is a tensor and its structure and details are discussed in subsection 3.2.

The centralized DNN is a machine learning model consisting of series of multiple 3D-CNN layers (represented by parameters θ_{CNN}), which the first one has an input tensor $\boldsymbol{d}_{t,N}$ described above, and after convolutional layers series of fully connected layers (FC, respresented by parameters θ). Number of layers of each type can vary based on the complexity of the of the predicted problem, but further considered model has two 3D-CNN layers chained after each other followed by two FC layers, where there are ReLU functions inserted after each layer (3D-CNN or FC) for non-linearities and after that there is one FC layer which produces an output $\hat{y}_{t+1}$. Figure 1 shows a centralized DNN architecture overview featuring on specific instance of centralized DNN following definition 3.1.

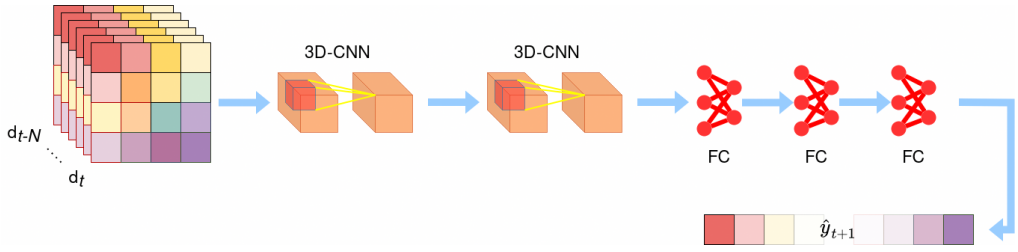


Figure 1: Centralized DNN architecture overview featuring sample processing for getting allocation in next time step.

Definition 3.1 (Centralized DNN)**.** Lets have parameters of two 3D-CNN layers $\theta_{CNN,1}$ and $\theta_{CNN,2}$ and three FC layers with parameters θ_1, θ_2, θ_3. Lets also have an *input* sample $\boldsymbol{d}_{t,N} \in \mathbb{R}^{H \times W \times N}$. A function producing an output

$$\hat{y}_{t+1} = \mathcal{DNN}(\boldsymbol{d}_{t,N}, \theta_{CNN,1}, \theta_{CNN,2}, \theta_1, \theta_2, \theta_3) \tag{1}$$

is called *centralized DNN* and the output $\hat{y}_{t+1} \in \mathbb{R}^M$ *allocation* if the function combines layers in consecutive manner. Last hidden FC layer and have applied a dropout of probability p.

Training of this centralized DNN is as standard regression machine learning model and similar structure has been offered by [5]. The most important contribution its authors have shown is the asymmetrical objective function they have used for training their model as described next.

Now, lets think about allocation as a provision of a resource and denotes under provision as u and over provision as o. Both are binary functions which are equal to one if their corresponding even occurs and and zero otherwise. Then lets denote resource allocation offered by some prediction model $\hat{y}_t$ at the time t. An objective function $f(\cdot)$ is a function assigning some penalty p to allocation $\hat{y}_t$ - actual traffic demand $y_t = d_t$ difference and can be written as follows:

$$f(\hat{y}_t) = p(\hat{y}_t - y_t) = up_u(\hat{y}_t - y_t) + op_o(\hat{y}_t - y_t), \tag{2}$$

where second equality shows overall penalty assigned by objective function as a sum of over and under provision penalties. The objective function proposed by authors of [5] corresponds to a financial side of resource allocation where authors argue that in field of telecommunications if users experience disruption of a service due to resource under provision the network operator has to pay some fixed penalty imposed by the SLA. But on the other hand, if service resources are over provisioned operator loses money proportionally by resource wasting. That can be written in previously offered notation as:

$$f(\hat{y}_t) = up_c + op_o(\hat{y}_t - y_t), \tag{3}$$

where p_c is constant penalty of SLA violation. Later will be shown that this is not the only objective function leading to good performance, but it yields good results if combined with above described models. Different SLAs can have different penalties for SLA violation, but it is not important here and will be omitted in further text.

To sum up the centralized DNN used in this work it follows definition 3.1 and is trained using objective function offered in [5].

3.2 Data Preprocessing

As it was already mentioned above, machine learning model described and used in this work uses 3D-CNN layers for extraction and recognition of features and patterns in the traffic history. The input as also described above is a 3D tensor $\boldsymbol{d}_{t,N}$ of traffic load for multiple time instances. Since convolutional layers extract information about correlation in the input, as it is the case in picture classification in 2D variation, the input has to be formed in a way layers can work with. That means time series of each VNF can not be simply put together. It is actually convenient to form them in a tensor where one dimension corresponds to time and other two form a 2D grid where each element is a one VNF and VNFs in the grid are placed

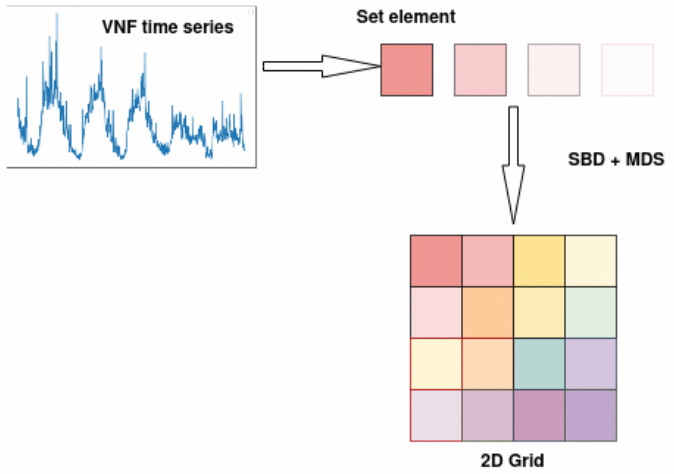


Figure 2: VNF time series placement in 2D grid. Multiple time series are aligned together in a pattern generated by minimization technique reflecting time-frequency features of each VNF transformed into bicoordinations.

next to each other if their time series correlate more then others. For better idea lets have a 2D grid and element $\mathcal{A}$ corresponding to some VNF. This elements is at the arbitrary spot in the grid. Then lets have another two elements $\mathcal{B}$ and $\mathcal{C}$ both corresponding to some VNFs. Then if $\mathcal{A}$ correlates to $\mathcal{B}$ more then $\mathcal{A}$ to $\mathcal{C}$ than $\mathcal{B}$ will be placed closer in the grid to $\mathcal{A}$ than $\mathcal{C}$, Figure 2.

In detail, idea is taken from [5], there is again a set of $\mathcal{M}$ VNFs and their corresponding time series where now considered in its whole measured history up until time t. Each of the time series might have different scaling - one VNF can experience much higher traffic than other and this might cause a bias towards the VNF with higher traffic, thus all time series have to be normalized. There are several versions of normalization, but there will be just one considered here which is called MinMax scaling (other example might be mean normalization). MinMax scaling [12] works as follows:

$$x_s(t) = \frac{x(t) - x_{min}}{x_{max} - x_{min}}, \tag{4}$$

where x is s time series, x_s is normalized version of x and x_{min} resp. x_{max} is minimum value of x, maximum respectively. All used time series are normalized using this technique and in further text mentions about it are omitted.

Further step is to determine level of correlation between time series. For this purpose there are multiple metrics which can determine time and/or frequency similarity of two time series like Dynamic Time Warping (DTW) [13] or Shape-Based Distance (SBD) [14]. The SBD is state-of-the-art metric for this purpose (known at the time of writing). Applying SBD to each pair of time series $(x_i, x_j), i, j \in M$ will create a cross-distance matrix $Q \in \mathbb{R}^{M \times M}$ with elements $q_{ij} = SBD(x_i, x_j)$.

Determining similarity between each VNFs has to be then transformed to 2D grid coordinates. Than means that the ultimate goal is to assign a pair of coordinates. Thus now, it has to be applied an algorithm able to get a set of coordinates in certain space (in this case 2D space) from cross-distance matrix, or cross-correlation matrix in general. This algorithm is Multi Dimensional Scaling (MDS) [15] [12], which is

formulated as minimization problem:

$$\min_{c_1,...,c_M} \sum_{i<j} (||c_i - c_j|| - q_{ij})^2, \tag{5}$$

where $\{c_1, ..., c_M\}$ is a set of 2D coordinates. MDS tries to minimize euclidean distance of two elements under constraints of cross-distances. After finding these coordinates they have to be transformed into regular grid, which can be solved by using linear assignment [16] implemented in [17]. In general 2D grid does not have to be a square but any rectangle with M elements and shape $H \times W$ is possible.

Finally to create set of input samples of set $\mathcal{M}$ VNFs for the model at time t will lead to a tensor with shape $H \times W \times N$ where each time series $d^i_{t,N}$ is scaled by its own minimum and maximum and placed between other time series in above described way.

Thus, there has to be a difference between training and runtime and it will be explained on the data which have been used for training and evaluation of this model. Used data are part of [18] and only a portion was used which is just an *internet* traffic served by some subset of BS. Maximum time series length is 3024 time steps and even though not all BS time series has full length only those with full length are used. Work considers multiple different sets of BS with cardinality $M = 16$, even though cardinality can be equal to any number enabling set $\mathcal{M}$ be formed in 2D grid. The same as the cardinality M the window has always the same length $N = 6$ in this work.

All time series set $\mathcal{M}$ is divided in three part for training, validation and testing purposes. Training uses the first half of all time steps and the second half is further divided into two equal parts where the first of them by time is assigned to be a validation and the second to testing dataset, or in this subsection, partition to distinguish dataset $\mathcal{M}$ including all time steps of the time series.

All partitions has to be reordered in the same way, but only two partitions (training and validation) are known to a potential operator at the time of the training, thus, only these two are used for scaling. These two partitions are used to get minimum and maximum for each time series and later for creation of cross-distance matrix. After normalizing two first partitions using (4) testing partition is scaled by using minima and maxima in (4) previously got from the first two partitions. As for the last step of the input forming the first two partitions are also used to obtain 2D grid coordinates as described above, where 2D grid coordinates are the same for each time series in testing partition as they are assigned to training and validation partitions.

After creating partitions, normalization with scaling and grid coordinates assignment each partition is used for creation of samples which the DNN can process - tensors with shape $H \times W \times N$. In this work 2D grid shape is square with $H = W = 4$.

3.3 Solution Architecture

As already mentioned above, the centralized DNN is computationally heavy, thus, it has to be located in the cloud or some network node with lot of computational power. It also needs to gather all its input information from all over the network.

To create a parallel here one can imagine again a set where operator wants to ensure that base stations in its mobile network will serve users adequately. A prediction model, the DNN, will then be placed in the MEC. If the DNN is centralized

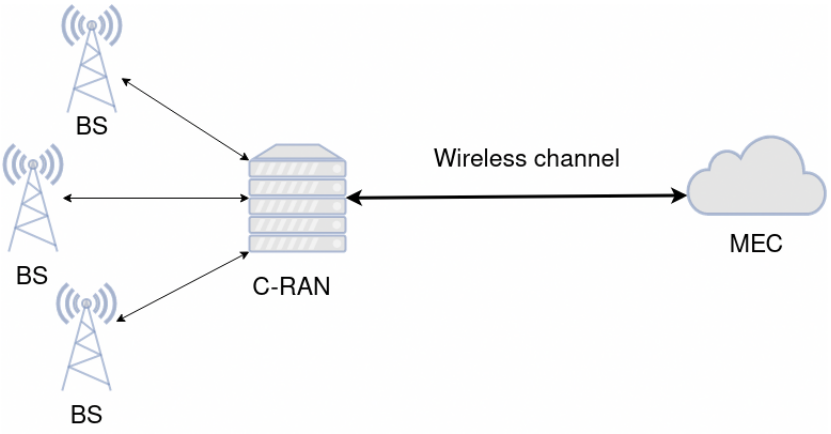


Figure 3: Basic scheme of the telecommunication network part harboring the described system. In case without slices each base station needs allocation and returns to C-RAN actual user traffic demand, then MEC gets aggregated passed traffic demand based via wireless channel.

then history of traffic for each BS of last N time steps has to be sent to the MEC (or just the last one if previous $N-1$ time steps are stored at the MEC). Thus, C-RAN has to sent this information over the network, then the MEC has to send its allocation back over the network again. The DNN can not be run on C-RAN due to computational constrains. But C-RAN has some additional capabilities which can be utilized. Architecture basic scheme is provided in Figure 3.

Authors of [6] have shown in their work a way how to distribute a DNN model consisted of multiple layers over several nodes where all nodes are chained in the same manner their layers are in equivalent centralized model, each node has an input as output from the last layer of its predecessor or the original input and node output is always (except the last node) consisted of its last layer output and an *exit*. An exit is a layer or series of layers transforming the last node layer output into the DNN output.

The DNN model used in this work can be spread, according to parallel with BS provision, across two entities in the telecommunication network a C-RAN and a MEC, which both serve as the model node and the first node is C-RAN which accommodates the first 3D-CNN layer of the centralized DNN followed by ReLU function. The exit of this node is called *Local exit* and consists of one hidden FC layer followed by ReLU function and the output layer. Its convolutional layer has a stride and a padding, both equal to one and a cube kernel with size equal to three.

Definition 3.2 (Local exit)**.** Lets have 3D-CNN layer θ_{CNN_L} of some kernel, stride and padding of arbitrary values. Lets also have two additional FC layers θ_1 and θ_2, both having their respective numbers of neurons. For all layers θ represents the network parameters of *weights* and *biases*. Then a *local exit* is a function $\mathcal{DNN}^l$ for input $\boldsymbol{d}_{t,N} \in \mathbb{R}^{H \times W \times N}$:

$$\hat{y}^l_{t+1} = \mathcal{DNN}^l(\boldsymbol{d}_{t,N}, \theta_{CNN_L}, \theta_1, \theta_2), \tag{6}$$

where the output $\hat{y}^l_{t+1} \in \mathbb{R}^M$ is *allocation* if the function combines layers in consecutive manner. Part of the local exit is 3D-CNN layer θ_{CNN_L} which takes as an input sample $\boldsymbol{d}_{t,N}$ and produces layer output $m_{t+1} \in \mathbb{R}^{f \times h \times w \times n}$, where $f > 0, f \in \mathbb{N}, 0 < h \leq H, 0 < w \leq W, 0 < n \leq N$, f is number of filters of the 3D-CNN.

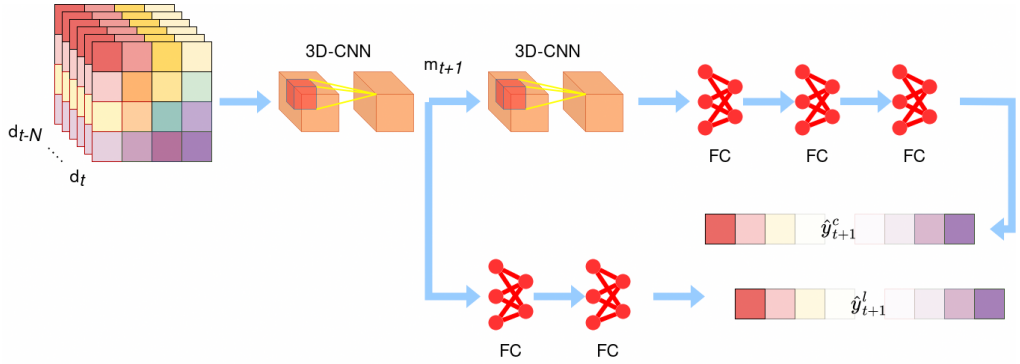


Figure 4: Overview of suggested DDNN architecture. Architecture demonstrates creation of two allocations from a sample for both local and remote exit.

The number of filters of the first 3D-CNN layer is $f = 32$ for this work. The hidden layer in the local exit has 32 neurons where the last layer has 16 neurons.

The second node consists of one 3D-CNN layer followed by ReLU function, two hidden FC layers, also followed by ReLUs and an output layer. Its exit is called *Remote exit*. The convolutional layer of the remote exit has stride equal to one, padding equal to two and cube kernel of size five.

Definition 3.3 (Remote exit)**.** Lets have 3D-CNN layer θ_{CNN_C} of some kernel, stride and padding of arbitrary values. Lets also have three additional FC layers θ_1, θ_2 and θ_3, all having their respective numbers of neurons. For all layers θ represents the network parameters of *weights* and *biases*. Then a *remote exit* is a function $\mathcal{DNN}^c$ for input $m_{t+1} \in \mathbb{R}^{f \times h \times w \times n}$, where $f > 0, f \in \mathbb{N}, 0 < h \leq H, 0 < w \leq W, 0 < n \leq N$:

$$\hat{y}^c_{t+1} = \mathcal{DNN}^c(m_{t+1}, \theta_{CNN_C}, \theta_1, \theta_2, \theta_3), \tag{7}$$

where the output $\hat{y}^c_{t+1} \in \mathbb{R}^M$ is *allocation* if the function combines layers in consecutive manner.

The 3D-CNN layer in the remote exit produces $f = 16$ filter. The first hidden layer of the remote exit has 64 neurons, the second hidden layer has 32 neurons and the output layer 16 neurons.

Both exits are capable of producing resource allocation for the whole set of base stations in the next time step. The important difference between allocations of different exits is that local exit is produced by network with lower capabilities and prediction power. That makes their allocation also different and theoretically allocation created by remote exit should be always more accurate due to additional capability of the network it is used to produce it. In practise it is not always the case but generally it is the idea why to use this kind of distributed architecture for the DNN.

Theoretically DNN can use local exit for production of its allocation in every time step and only when it decides allocation accuracy is not sufficient it can send first node layer output to the remote node to obtain more accurate allocation. The section 5 describes closely how to decide about whether to use local or remote exit allocation and also subsection 3.5 presents results with using one of decision processes how to obtain optimal trade off where portion of allocation is served by local exit with sufficient accuracy and the rest is served by remote allocation which lowers the communication need.

3.4 Distributed Training and Runtime

The biggest challenge for this kind of machine learning model distribution is a question how to train such a neural network. This work utilize two phase training mechanism consisting of forward pass yielding a DNN output and its loss respectfully and back propagation. The back propagation passes the output loss through the neural network using a gradient descent. In this work the optimizer or the stochastic gradient descent variant is always Adam [19].

Algorithm 1 DDNN training step

Require: $\boldsymbol{d}_{t,N}$, y_{t+1}, w_l
Ensure: $\mathcal{L}_t$
$\hat{y}^l_{t+1}, m_{t+1} \leftarrow \mathcal{DNN}^l(\boldsymbol{d}_{t,N})$
$\hat{y}^c_{t+1} \leftarrow \mathcal{DNN}^c(m_{t+1})$
$\mathcal{L}^l_t \leftarrow w_l f(\hat{y}^l_{t+1} - y_{t+1})$
$\mathcal{L}^c_t \leftarrow (1 - w_l) f(\hat{y}^c_{t+1} - y_{t+1})$
$\mathcal{L}_t \leftarrow \mathcal{L}^l_t + \mathcal{L}^c_t$

This setting is trying to create a prediction mechanism, thus, it is a regression problem. Centralized DNN is trained that at every time step $t-1$ the DNN is fed a sample with structure described in subsection 3.2, creates an allocation $\hat{y}_t$, in forward pass, which compares with actual traffic demand y_t and by applying objective function to the allocation error $e_t = \hat{y}_t - y_t$ produces an allocation *loss* (algorithm 1) which is used by *optimizer* for updating neural network weights and eventually *learn* the regression for resource allocation problem.

Distributed DNN has several outputs which each of them has different allocation error or loss respectfully. [6] suggested to create a combined loss of all allocation errors by summing weighted losses of all exits. Then DDNN has one scalar loss value which can be used for back propagation because weighted sum has a derivative. Weights of each exit loss, in the case of two exits - local weight w_l and cloud weight w_c, can be seen as an importance ratio in the overall performance. It means if the weights are both equal to the same value then both of them have the same importance. Also, if $w_c = 1$ and $w_l = 0$ the DDNN is equivalent to centralized DNN because local exit does not contribute in back propagation.

Algorithm 2 Model run time forward pass

Require: $\boldsymbol{d}_{t,N}$
Ensure: $\hat{y}_{t+1}$
$\hat{y}^l_{t+1}, m_{t+1} \leftarrow \mathcal{DNN}^l(\boldsymbol{d}_{t,N})$
Model state $\leftarrow (\boldsymbol{d}_{t,N}, \hat{y}_{t+1})$
Decision $\leftarrow$ Decision process(Model state)
if Decision is *local* **then**
    $\hat{y}_{t+1} \leftarrow \hat{y}^l_{t+1}$
else if Decision is *remote* **then**
    $\hat{y}^c_{t+1} \leftarrow \mathcal{DNN}^c(m_{t+1})$
    $\hat{y}_{t+1} \leftarrow \hat{y}^c_{t+1}$
end if

It can be seen that the most important are not actual values of weights but their ratio. Additionally, it can be observed that significance of the local exit should

not be low because it still has to produce meaningful result and it is already under powered neural network. Further, [11] suggested that earlier exits should be given higher significance, because their performance also improves performance of later exits by providing additional regularisation, this is a premise more discussed in subsection 5.4. Taking all notes into account weights for two exits can be simplified in a single real scalar local weight $w_l \in [0, 1]$ and remote exit weight is always taken as $w_c = 1 - w_l$.

Loss function for the distributed DNN can be formalized as:

$$\mathcal{L}_t(e_t) = \underbrace{w_l f(e_t^l)}_{\text{local exit loss } \mathcal{L}_t^l} + \underbrace{(1 - w_l) f(e_t^c)}_{\text{remote exit loss } \mathcal{L}_t^c}, \tag{8}$$

where $f(\cdot)$ is an objective function and e_t^l is an local exit allocation error, e_t^c remote exit allocation error respectively.

By the standard training mechanism after forward pass and counting a loss of the DNN output the neural network weights are updated by using standard back propagation via Adam optimizer.

The same as the training phase runtime phase also uses forward pass of the DDNN, but is is a bit different, algorithm 2. The DDNN input as it is described in subsection 3.2 is fed to the DDNN. The local exit produces its allocation $\hat{y}_{t+1}^l$ that alongside with input $\boldsymbol{d}_{t,N}$ is evaluated by a decision algorithm, better described in section 5, producing decision whether to yield local exit allocation as the overall output or the allocation should be done by remote exit. If the later is true, then local layer output is *send* to remote node via network where remote exit produces the overall output.

3.5 Evaluation of Distributed DNN

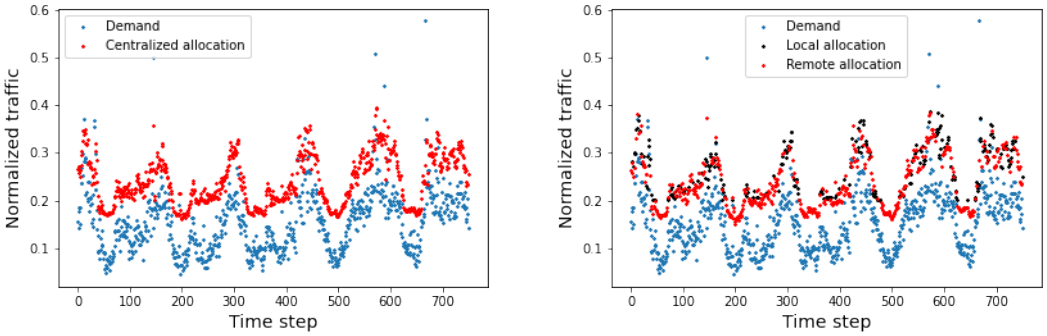
To prove an added value when the distribution DNN is used instead of the centralized DNN architecture it can be just stated that the distribution is a benefit itself and then it can be just shown to what extant DNN performance is affected by the distribution.

Benefits and consequences of distribution is hard to evaluate in terms of computational power level saved. But one aspect of distribution can be evaluated quantitatively - the communication overhead O. By simple stating that if centralized DNN is used the network would have to bear some number of packets sent and the improvement uses just a fraction of them expressed in percentages. Centralized DNN or its equivalent distributed version have both $O = 100\%$ and DDNN having all allocations made by the local exit has $O = 0\%$.

Actual performance change can be evaluated rather easily. Every allocation already has its metric for training purposes in form of loss function or objective function, thus, performance can be expressed in the change of allocation loss.

This subsection considers two DNNs, first *centralized* DNN and second *distributed* DNN, both described above. It shows and compare their performance in accuracy of allocation measured by loss function $\mathcal{L}_n$ (cumulative loss normalized per sample) and it communication overhead O.

Both DNNs use previously described architectures and as objective function take a function closely described in subsection 4.1 as *additive* objective function with penalty (14) suggested by [5]. This function has two parameters, which are set here



(a) Centralized model allocation

(b) Distributed model allocation

Figure 5: Allocations with respective actual traffic demand of one base station id 903 in the set created by centralized and ideal distributed model, which has also distinction in exit producing allocation at the time step.

to be learning slope $\epsilon = 0.1$ and under provision cost penalty $\alpha = 0.5$, which is also denoted in this work as p_c.

$$\mathcal{L}_n = \frac{1}{T}\sum_{t=0}^{T}\mathcal{L}_t = \underbrace{\frac{1}{T}\sum_{t=0}^{T}\mathcal{L}_t^l}_{\mathcal{L}_n^l} + \underbrace{\frac{1}{T}\sum_{t=0}^{T}\mathcal{L}_t^c}_{\mathcal{L}_n^c}, \tag{9}$$

where T is a number of time steps in dataset and $\mathcal{L}_t$ is taken from (8). Additionally, DDNN performance can add description on local exit level $\mathcal{L}_n^l$ and remote exit level $\mathcal{L}_n^c$.

Both DNNs are trained to create allocation for 16 time series taken from dataset [18] with 16 consecutive base station identification numbers "bs_id" starting at id 900, later denoted as dataset A. These time series were normalized, split in three partitions/datasets and organized as described in subsection 3.2.

Both DNNs trained and evaluated with using testing part of data produce allocations shown by Figure 5. Distributed DNN trained with the local exit weight $w_l = 0.7$ produces similar allocation as centralized DNN $w_l = 0$. It is hard to see difference in Figure 5, but allocations performance comparison is in Table 1.

Table shows values of normalized loss per sample $\mathcal{L}_n$ by which performance of the DDNN is actually better (lower loss) than centralized DNN. It can happen and is better described in [6]. That means that certain number of allocations created by the local exit have to be actually better (again have lower loss) than the remote one.

Final allocation for DDNN is the loss lower bound in this case, that means that both exits produce their allocations and respective losses, which are compared and final allocation is taken as the one correspond with the lower loss. Ideal decision, or better call it *a posteriori* decision, has to be substituted for decision process, section 5, for use in practice during runtime.

Table 1: Performance comparison showing effect of DNN distribution.

Model	$\mathcal{L}_n$	$\mathcal{L}_n^l$	$\mathcal{L}_n^c$	O [%]
Centralized DNN	0.067	-	0.067	100
Distributed DNN	0.064	0.076	0.069	64.4

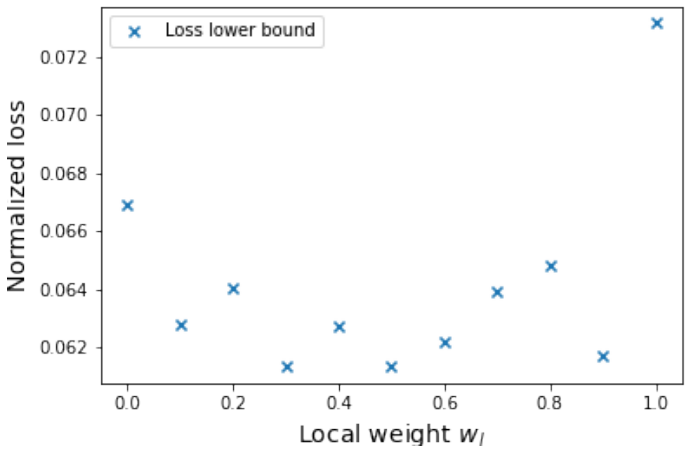


Figure 6: Normalized loss lower bound for DDNN trained for different local weights w_l. All values are obtained by evaluating trained DDNN using test dataset of BS id range 900 - 915 and objective function is additive, penalty (14)

Table 1 also shows that the communication overhead drops to $O = 64.4\%$. But it can be pressed even lower if there is a will to trade off performance for lower communication overhead, also shown in section 5 as penalized loss lower bound.

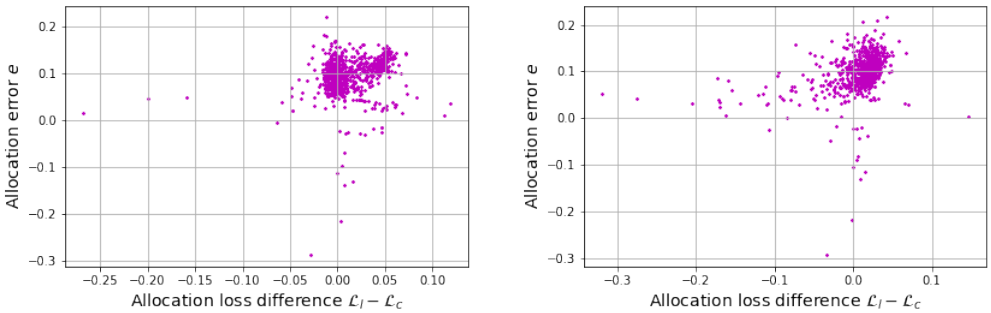
DDNN performance is local weight dependent as shown in the (8), where the dependency is shown for the loss function. But to show how this actually affects DDNN performance a function has to be shown: loss lower bound function of local weight. Figure 6 shows this function for normalized loss. The trend of the function is not clear due to evaluation on limited dataset. But what can be seen is that centralized DNN, which is marked in the figure as local weight $w_l = 0.0$ and purely local DNN, local weight $w_l = 1.0$, have the biggest normalized loss lower bound, proving that the distribution of DNN helps to increase overall performance and certain combination of local and remote allocations proves to increase DNN performance.

Definition 3.4 (Optimal allocation)**.** Lets have a certain DDNN trained for some local weight w_l. For each sample $\boldsymbol{d}_{t,N}$ DDNN can create two allocations: local allocation $\hat{t}^l_{t+1}$ and remote allocation $\hat{t}^c_{t+1}$. Each allocation has its allocation error $e^l_t = \hat{t}^l_{t+1} - y_{t+1}$, e^c_t respectively and using objective function with certain penalty policy each allocation is assigned loss $\mathcal{L}^l = f(e^l_t)$, resp. $\mathcal{L}^c = f(e^c_t)$. *Optimal* allocation is called the one allocation for which loss is lower then for the other allocation.

Definition 3.5 (Loss lower bound)**.** Lets have a certain DDNN trained for some arbitrary weight w_l. When the DDNN yields optimal allocation for each sample during slotted time period $T \in \mathbb{N}$ then mean loss over over time T of all DDNN optimal allocations are called *loss lower bound*.

Loss lower bound is the base case scenario and shows potential of the distributed DNN model performance. In practice this loss lower bound is hardly reached as is described in section 5, because there is no algorithm perfectly choosing allocation (local/remote) during run time.

For illustration Figure 7 shows relation between allocation error and allocation loss difference which is equivalent to loss lower bound. In this representation optimal



(a) Local weight $w_l = 0.1$ (b) Local weight $w_l = 0.9$

Figure 7: Relations between allocation errors and sample loss difference for DDNN trained for two local weights.

decisions are distinguished as two classes by the vertical line at $\mathcal{L}_l - \mathcal{L}_c = 0$.

As depicted in both plot in Figure 7 the relation between allocation loss difference and local exit allocation error is not entirely separable into two classes, which can lead to penalized loss lower bound defined in section 5 as a threshold for ideal decision process.

To conclude DDNN comparison with centralized DNN former DNN architecture has benefits which later lacks while maintaining performance abilities of centralized DNN. Also results show that DDNN architecture can be successfully used for regression problem and not only classification one as it is the case in original paper [6] of suggestion.

4 Slice Introduction

Network slicing is a concept suggested by 3GPP, for example in [7], and it is a way how to create a virtual network with specific features or certain geographical area in a network defined on top of some physical infrastructure. A network slice has to seem to its users or more precisely UEs belonging to that peculiar network slice as a fully functional, distinguishable and standalone network.

Still the network slice is a part of the broader network operated by some operator. This operator needs to be able to operate the whole network while serving this standalone slice. Operator network can potentially support multiple slices, which might have to share physical resources with each other.

For better idea lets think that operator has some part of the spectrum to its disposal over its whole geographical area it operates. Then two customers of the operator wants to have their own slices (one slice for each customer). These two customers operate and want their slices to be operational over the whole geographical area the operator operates in (for example one country or for better understanding one island). In that case operator must be able to serve both customers simultaneously and it can create a subdivision of its spectrum and serve each customer with one partition of the spectrum, described as reserved base approach [8]. This is valid a possible scenario but it limits bandwidth for each customer, also described as lack of scalability.

The operator might actually try shared base approach [8], to predict which slice needs how big chunk of the spectrum in some area served by some base station and serve one customer more that the other in this area if it has a need for more bandwidth than the other customer. This need might flip in the next time step and the second customer might need more than the first one.

A situation such as this one can happen and can be even more complicated, but operator can deal with the situational needs if it can plan them. Lets think about bandwidth as a resource, it is limited but for simplicity lets not put this constraint in the system so far. Then the operator network has base stations which serve at the whole time both customers. Since both customers live in their respective slices their provisions for each base station in the operator network can be described separately but their are not independent to each other (they might share users with UEs for both slices, one BS operates in the same area for both slices, etc.). Thus, one slice resource, bandwidth, can be predicted as described above. But there is also potential benefit from observing slice of second customer while creating an allocation for slice of the first customer, since they are not independent to each other.

When each BS operates multiple VNFs allocating common resource, in the rhetoric of previous example bandwidth, for different slices every VNF can be described by its history, allocations at every time step. Those VNFs from all slices can be seen as before as time series of resource provision and the whole set of VNFs, let say again of $\mathcal{M}$, can be predicted jointly as described in section 3.

Prediction made in the same manner as in the section 3 ensures any VNF in the set is served adequately, but it does not say anything about slices. This section describes two possible ways how to take a slice into account in prediction process using objective function for training machine learning model. It also discuss benefits of each approach and at the end it offers performance comparison of suggested objective functions shown on example allocations.

4.1 Additive Objective Function

Objective function enforces certain goals during training of the machine learning model and ensures by that seek behavior during prediction time. Additive objective function seek to ensure that certain policy like that suggested by authors of [5] (3) is enforced for every VNF in model.

Definition 4.1 (Policy and Objective)**.** Lets function assigning a loss or penalty p to one VNF error e_t^i in DNN allocation call *policy* and denoted $p(e_t^i)$ and function combining these penalties in one single scalar loss or cost of the whole allocation $\hat{y}_t$ call *objective* function $f(e_t)$.

Definition 4.2 (Additive Objective)**.** An objective function which can be formulated as:

$$f(e_t) = \sum_{i \in \mathcal{M}} p(e_t^i) \tag{10}$$

is called *additive* objective function.

It can also be scaled, for example by factor $1/M$, which assigns normalized penalty to allocation as its loss. Unscaled formula represents cumulative penalty if policy penalizes one VNF by money it wastes by SLA violation or proportional over provision then objective function has a meaning of total amount of money wasted or additional cost to operational costs for operator. This work favors scalar $1/M$ simply for better notion on how well is each VNF predicted in allocation.

This objective is applied when there is no connections between VNFs in terms of slice or can be seen as objective function ensuring allocation for one slice.

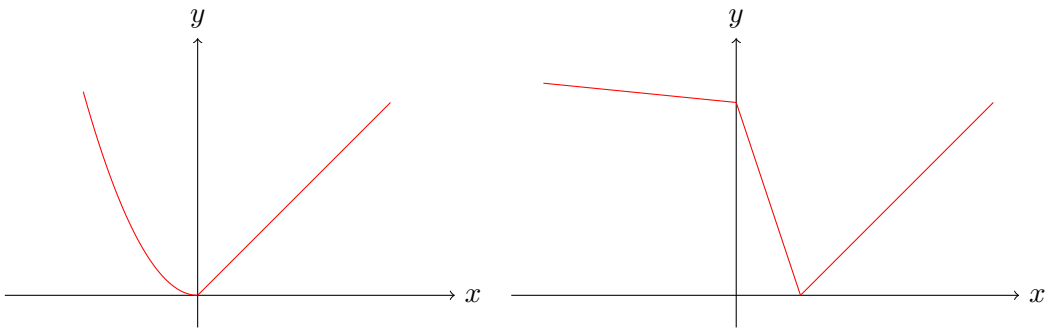
4.2 Coupling Objective Function

SLA for normal customer with single UE is different from the SLA for a network slice. Single user SLA is usually violated when user experience service interruption. But user can not be at the same time served by all base stations in the network, thus, if some base stations are under provision user does not experience interruption in service if the user is served by some other BS.

On the other hand, network slice experiences service interruption if any part of it experiences it. Thus, SLA is violated when ever any VNF used for serving network slice is under provisioned. Any objective function reflecting slicing in the network has to take this into account. There is surely multiple ways how to enforce penalization of allocation if any part of slice experiences under provision, but this work considers only one.

Definition 4.3 (Coupling Objective)**.** Lets have model allocation $\hat{y}_t$ with error e_t and for set of time series $\boldsymbol{d}_{t,N}^i, i \in \mathcal{M}$ (VNFs). Also, lets any arbitrary subset of time series $\boldsymbol{d}_{t,N}^i$ call slice $\int \subset \mathcal{M}$. Each time series belongs to only one slice and number of slices is denoted S, cardinality of $\int$. Each slice s_j is assigned a penalty:

$$p_t^j = \begin{cases} p\left(\max_{i \in s_j}\{e_t^i\}\right), & \forall e_t^i \geq 0 \\ p\left(\min_{i \in s_j}\{e_t^i\}\right), & \exists e_t^i < 0 \end{cases}. \tag{11}$$



(a) Quadratic under provision (13) (b) SLA violation policy (14)

Figure 8: Illustrations of penalty policies functions.

An objective function counted as:

$$f(e_t) = \sum_{j \in \mathcal{S}} p_t^j \tag{12}$$

is called *coupling* objective.

The reason why this objective function is called *coupling* is because when the objective was firstly used it grouped only pairs of time series, but the name persevered even after the change which enabled use of any arbitrary grouping.

The main idea behind the coupling objective function is to penalize the whole slice by the penalty assigned to the worst predicted time series. The *worst* performing prediction of time series has to be also defined, but in this case it split to two cases. If any of time series is under provision by the allocation the whole slice is penalized as this under provisioned time series, or as the time series which is the most under provisioned. If there is no under provision time series in allocation then the whole slice is penalized as the most over provision time series in the slice.

4.3 Penalty Policies

There is a possibility to describe several ways of penalization for single network element, either VNF or slice. One possible policy is the loss function described above (3). Another possible policy, which is used in this work as well, can be:

$$p(e_t) = uce_t^2 + oe_t, \tag{13}$$

where can be seen that under provision is penalized more strictly than over provision. If the error is very small, $|e_t| < 1$, under provision is actually penalized less than over provision, thus, equation has also constant c which should be some reasonably large for eliminating this problem. For this work $c = 50$.

Penalty policies are used for training as part of the loss function, thus, they have to be continuous and differentiable and derivative has to be finite and non-zero. Differentiability is needed for gradient descent learning algorithms and finite and non-zero values are needed for convex optimization for minimum search.

Second policy (13) meets all conditions, but first policy (3) is not continuous at $e_t = 0$ and its derivative for all under provision part is equal to zero. Authors of [5]

suggested to create the function in following manner:

$$p(e_t) = \begin{cases} p_c - \epsilon e_t, & u = 1 \\ p_c - \frac{1}{\epsilon} e_t, & o = 1 \wedge e_t < p_c \epsilon \\ e_t - p_c \epsilon, & o = 1 \wedge e_t \geq p_c \epsilon. \end{cases} \tag{14}$$

There can be other policies defined, but as said earlier it has to penalize under provision more than over provision to fulfil the goal of allocating at least as needed amount of resources.

4.4 Performance Comparison

Additive objective is reliably working as illustrated not only in subsection 3.5. Coupling objective seeks different goal but it has to be compared to previous results or method, in this case additive objective function to show that the performance of new method is at least the same and not worse. The goal of the comparison is to show desirability of the coupling objective function. It is important to apply the coupling objective function in some example case and evaluate that the allocations produced while using this objective function are qualitatively good as prediction of the traffic and also that the objective meets its goal better than previously used additive objective.

This subsection compares additive and coupling objective functions as follows:

1. each objective function is used to train the same centralized DNN subsection 3.3, then
2. trained models produces allocations for the same test set of samples,
3. their produced allocations are evaluated using the coupling objective function.

Definition 4.4 (Allocation comparison)**.** Whenever one set of time series is predicted two allocations $\hat{y}^1_{t+1}$ and $\hat{y}^2_{t+1}$ at time t and they are compared with the set actual traffic values y_{t+1} using one objective function f, then it is said that $\hat{y}^1_{t+1}$ is *better* allocation than $\hat{y}^2_{t+1}$ if and only if:

$$f(\hat{y}^1_{t+1} - y_{t+1}) < f(\hat{y}^2_{t+1} - y_{t+1}), \tag{15}$$

otherwise if these two values are equal allocations are called to be *equivalent* and in the last case allocation $\hat{y}^1_{t+1}$ is called *worse* than allocation $\hat{y}^2_{t+1}$.

If the DNN trained with coupling objective function yields significantly better results then the other DNN, it can be said that coupling objective function is better to used for resource allocation problems with slice management taken into account.

Other side of the coin can be if DNN trained with coupling objective function yields significantly worse allocation then the other DNN, then it can be said that the coupling objective function is not usable in the form previously described subsection 4.2 for resource allocation. In that case other function has to be formulated and compared to prove its desirability.

There is also a third possible option of comparison - same results. It may also be that allocations would be close to each other, but due to uncertainty it would not be possible distinguish two extreme cases where one is better than the other.

Definition 4.5 (Objective function comparison)**.** Lets have two objective functions f and g. Lets have series S of samples $\boldsymbol{d}_{t,N}, \forall t \in [0,T]$ created on top of set of time series $\mathcal{M}$ with the window length N. A DNN model structure once trained using objective function f and call this trained model $\mathcal{F}$ and once trained using objective function g and called this trained model $\mathcal{G}$ can be compared in performance. When both models predict sequences of allocations $\mathcal{F}(S)$, resp. $\mathcal{G}(S)$, using S which both are evaluated by objective function f then if:

$$|f(\mathcal{F}(S)) - f(\mathcal{G}(S))| < T\epsilon \tag{16}$$

objective functions are *equivalent* for the error ϵ.

Coupling objective function needs as a parameter certain slice pattern - list of slices with their respective VNFs. Performance of the coupling objective can be dependent on the slice pattern, thus, coupling objective has to be compared for several instances of the pattern.

Definition 4.6 (Slice pattern)**.** Lets have a set of time series (VNFs) $\mathcal{M}$ organized in the 2D grid (subsection 3.2). Each time series has assigned an index i_{idx} in the grid based on its grid coordinates (i, j): for example time series has assigned index 0 if its coordinates are (0,0) and 1 if its coordinates are (0,1), $i_{idx} = j + iW, \forall i \in [0, H], \forall j \in [0, W], H \times W = M$. Lets have set of slices $\mathcal{S}$ with indices s_{idx}, then *slice pattern* is a function mapping indices of time series in the grid to indices of slices and can be written: $\{s_{idx} : \{i_{idx}\}\}$.

This subsection considers three slice patterns (named):

- Pairs: {0:{3, 6}, 1:{1, 2}, 2:{12, 14}, 3:{4, 11}, 4:{9, 15}, 5:{7, 10}, 6:{0, 8}, 7:{5, 13}},
- Equal slices: {0:{0, 3, 5, 6, 7, 10, 13, 15}, 1:{1, 2, 4, 8, 9, 11, 12, 14}},
- Unequal slices: {0:{0, 7, 10}, 1:{1, 2, 3, 4, 5, 6, 8, 9, 11, 12, 13, 14, 15}}.

These three cases of slicing are examples of two extreme cases (*Pairs* and *Equal slices*) and one regular case (*Unequal slices*). There is also one extreme case which is not considered - two slices where one of them consists of just one VNF and the other one includes all other VNFs of the set $\mathcal{M}$. But that case is excluded because it is a subcase of *Unequal slices*.

Machine learning models are also data dependent, that means to train really general DNN giving objective prediction it has to be trained with data which have no bias, under condition that the model is not overfit. Training datasets obtained from real data would most likely have some level of bias which may be hard to compensate. This bias can result in wrong evaluation of the model performance in the terms of objective functions comparison, thus, DNN has to be trained using several datasets to mitigate problem of the biased dataset.

Centralized DNN is trained, tested respectively, using three datasets created using technique described as training dataset, test dataset respectively, in subsection 3.2 and internet provision data in [18]:

- Dataset A: BS id range 900 - 915,
- Dataset B: BS id range 1050 - 1565,

- Dataset C: BS id range 3201 - 3216,
- Dataset D: BS id range 5004 - 5019.

Table 2: Objective functions performance comparison for centralized DNN trained on multiple sets {B, C, D} with different slice patterns by additive and coupling objective functions and evaluated by coupling objective function. Objective functions use the same penalty policy (14). Values in the table are cumulative objective function results for certain trained DNN and testing set normalized by length of the testing set - results are average sample loss assigned by coupling objective function to one slice in the set.

Case	Pairs		Equal Slices		Unequal Slices	
Set	Additive	Coupling	Additive	Coupling	Additive	Coupling
B	0.171	0.172	0.264	0.265	0.252	0.241
C	0.082	0.084	0.132	0.135	0.125	0.135
D	0.124	0.127	0.185	0.178	0.184	0.194

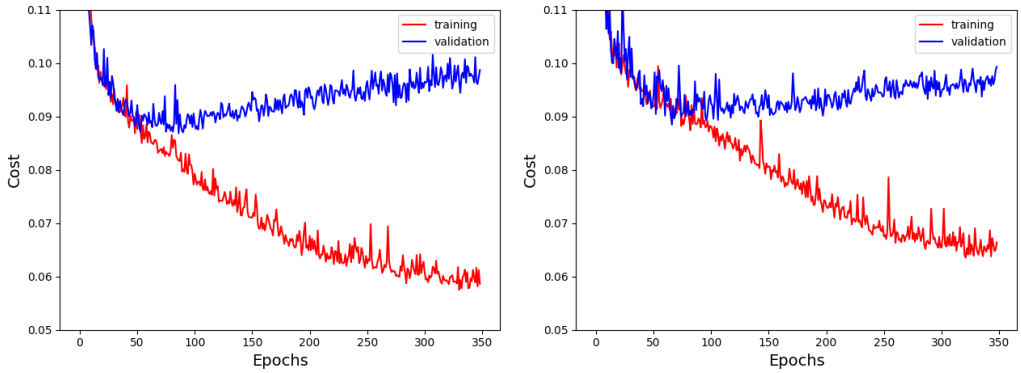
Objective functions comparison is shown in Table 2. Table values does not show any significant gain in using coupling objective function over additive objective function. In some slice pattern cases and data sets additive function is slightly better then the coupling objective function and for others the opposite is true. From that can be concluded that both objective functions are equivalent for the slicing objective because coupling objective function is not significantly better nor worse than the additive objective function.

This finding has a significant implications. First implication, can be said, is desirability of the coupling model objective. That means coupling objective meets its goal to provision each VNF to ensure its respective slice under provision cut and it still is able to sufficiently create overall VNF set allocation. Second implication is that additive objective function ensures similar overall performance for slice management of certain slice pattern as coupling objective function.

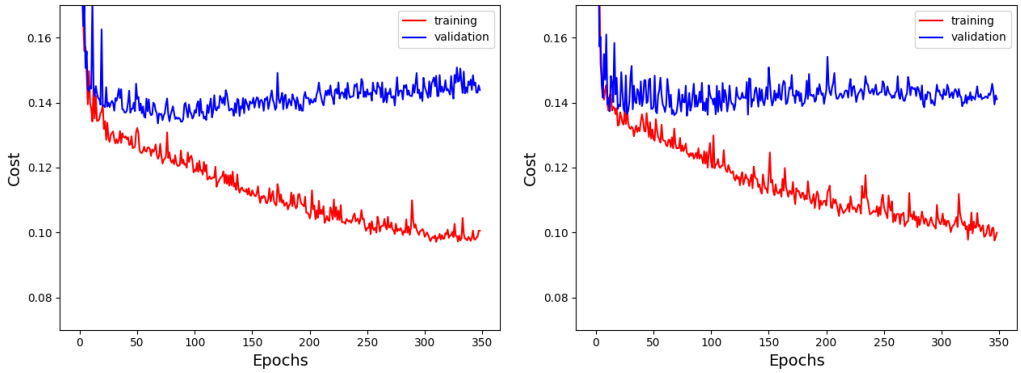
Algorithm 3 DDNN training step

Require: $\boldsymbol{d}_{t,N}$, y_{t+1}, w_l
Ensure: $\mathcal{L}_t$, $\mathcal{C}_t$
$\hat{y}^{l,\prime}_{t+1}, \hat{y}^{l}_{t+1}, m_{t+1} \leftarrow \mathcal{DNN}^l(\boldsymbol{d}_{t,N})$
$\hat{y}^{c,\prime}_{t+1}, \hat{y}^{c}_{t+1} \leftarrow \mathcal{DNN}^c(m_{t+1})$
$\mathcal{L}^l_t \leftarrow w_l f(\hat{y}^l_{t+1} - y_{t+1})$
$\mathcal{L}^c_t \leftarrow (1 - w_l) f(\hat{y}^l_{t+1} - y_{t+1})$
$\mathcal{L}_t \leftarrow \mathcal{L}^l_t + \mathcal{L}^c_t$
$\mathcal{C}^l_t \leftarrow w_l g(\hat{y}^{l,\prime}_{t+1} - y_{t+1})$
$\mathcal{C}^c_t \leftarrow (1 - w_l) g(\hat{y}^{c,\prime}_{t+1} - y_{t+1})$
$\mathcal{C}_t \leftarrow \mathcal{C}^l_t + \mathcal{C}^c_t$

Second implication is not actually very appealing outcome simply due to the fact that using the coupling objective does not gain any major performance benefit. More so when the time complexity of coupling objective function computation is higher that the additive one (simple comparison of mathematical operations needed to compute objective function).



(a) Training with set C with additive objective (b) Training with set C with coupling objective



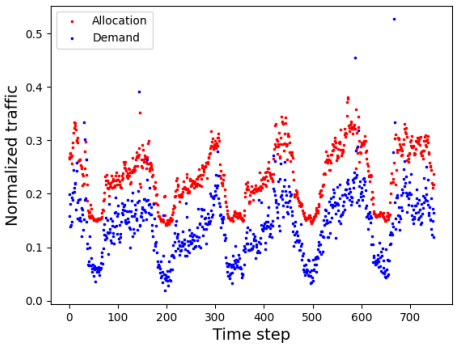
(c) Training with set D with additive objective (d) Training with set D with coupling objective

Figure 9: Training curves DDNN trained for additive and coupling objective evaluated by coupling objective function without dropout. Evaluation is done during training step.

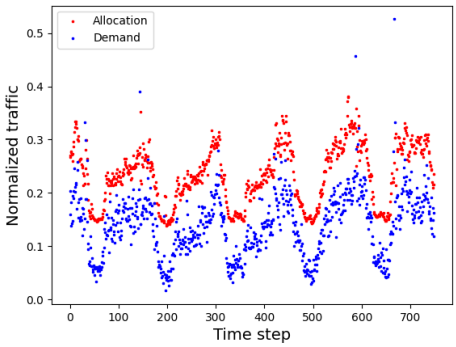
Table 2 shows another thing, overall performance drops significantly if slices contains a lot of VNFs. This is natural conclusion of provision penalty asymmetry and coupling objective to limit number of under provision VNFs.

In practice slicing management needs the ability to dynamically change VNF pertinence to a slice. It this case it is omitted for simplicity. But from results it can be seen that in practice it might not be easy to do, because change in slice pattern causes a change in the objective outcome so even though additive objective function is equivalent to coupling function with any slice pattern, they are equivalent only under evaluation of the coupling objective function. That means dynamic changes in slice pattern are possible but respective losses obtained by objective functions are not comparable.

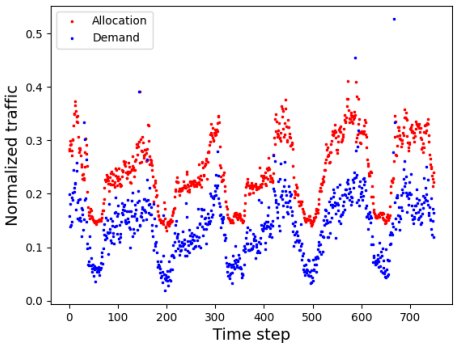
Values in Table 2 are subject to some error which is not estimated here. To offer one more example proving both objective being equivalent for slicing purpose it might be shown that in training DDNN converges towards the same optimum. The comparison of the training instances using different objective functions are done in training using step algorithm 3. The steps yields two losses at the end and for clarity the loss using intended objective function for training (in the algorithm denoted as f) is called training loss and still denoted $\mathcal{L}_t$. The other loss is loss intended for comparison between training instances and for distinction is called as *cost* $\mathcal{C}_t$ (and



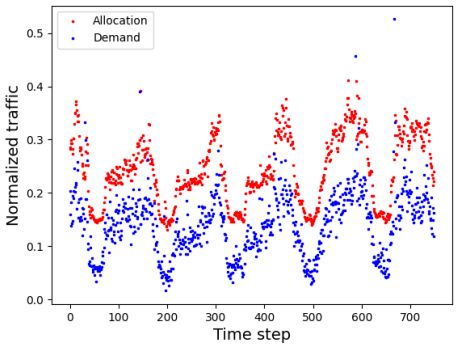
(a) Allocation created by additive objective function for BS id 3.



(b) Allocation created by additive objective function for BS id 6.



(c) Allocation created by coupling objective function for BS id 3.



(d) Allocation created by coupling objective function for BS id 6.

Figure 10: Allocations for one slice 0, two BS id 3 and 6 in sat A created by DDNN trained using additive and coupling objective functions. Plots include traffic demand for comparison.

uses coupling objective function g).

Training losses using two different objective functions in two training instances are incomparable. But when both training instances keeps track of the cost using the same objective function then results of these costs are comparable. Development of the training using two different datasets C and D is traced in Figure 9. All plots feature two training curves training phase cost development and validation phase cost development.

Optimal epoch is chosen using training loss, thus model optima might be slightly shifted from optima in the plots. Training loss function uses dropout with probability $p = 0.4$ after the last hidden FC exit layer while cost does not. This is described in algorithm 3 as $\hat{y}^{l,\prime}_{t+1}$ and $\hat{y}^{l}_{t+1}$, or $\hat{y}^{c,\prime}_{t+1}$, $\hat{y}^{c}_{t+1}$. The optimal epoch for the DDNN trained using additive objective function, as shown in Figure 9a, is epoch number 109 and its coupling counterpart, Figure 9b, is epoch number 146. Training instances using the dataset D, figures 9c 9d, have optima in epochs number 109 and 107 respectively.

Training curves includes training noise which might be reduced by increase of batch size. Actual batch size used in production all training instances in this work use 32 samples. The increase of samples in batch has also negative impact on training

convergence and this value have been chosen as the best to ensure rapid training.

DDNN allocations are created for each VNF in the set, thus even though the objective functions have equivalent allocations for each VNF in the set are different. There is a possibility that coupling objective function might create very accurate allocations for majority of the VNFs and significantly lack predictive capabilities for just a few VNFs in the set. That would be very unfortunate a significantly undermine possible use of the objective. Thankfully it is not the case and all VNFs are allocated evenly. Qualitative objective functions comparison for two cases of VNF allocations are shown in Figure 10.

To conclude a desirability of coupling objective. Coupling objective ensures its goals to enable DNN to create allocations for multiple slices at once. Also allocations yielded by DNN trained with this objective ensures equivalent overall performance for all slices as DNN trained with additive objective evaluated by the same objective ensuring the slices.

There have been only one suggestion to multi-slice objective function, but it is possible to show another. For example additive objective function, scaled by factor $1/M$, is a objective for one slice only, but this scheme might be applied to multiple slices environment as well, thus, another objective function might be:

Definition 4.7 (Multi-slice Additive Objective)**.** An objective function which can be formulated as:

$$f(e_t) = \sum_{j \in \mathcal{S}} p_t^j, \tag{17}$$

where $\mathcal{S}$ is a set of slices which include all time series of set $\mathcal{M}$ and for penalty policy p:

$$p_t^j = \frac{1}{s_j} \sum_{i \in s_j} p(e_t^i) \tag{18}$$

is called *multi-slice additive* objective function.

5 Decision Process

The subsection 3.5 shows the potential benefit of using distribution because some local exit allocations can be more accurate than the remote exit allocations and thus improve the overall performance of the DNN. The subsection 3.5 also shows the ideal choices of exit allocation by considering both local and remote allocations a posteriori. In practice this is not possible, because DDNN aims to avoid knowing the remote allocation. Thus, there is a need for an effective way how to decide whether to use local allocation (already known to the system) or the remote, unknown allocation. This decision has to be effective in recognising which allocations can be improved by remote exit and which can not.

In this work the distributed DNN regression model needs a reliable decision process for distinguishing cases in which local exit has sufficient allocation accuracy or not. Classifiers using DDNN have advantage over regression models due to entropy given to each classes with each sample. This basically means level of confidence the classifier has with its classification (taken as a maximum of classes entropy vector). This can be seen in [6]. For regression model there has to be some adequate substitution for entropy, because regression model does not have possibility to obtain entropy right out of the prediction.

Instead of entropy decision could be made based on an output of some algorithm able to evaluate complexity of prediction which have to be made about a sample for creating adequate prediction. This algorithm can either evaluate input sample, local exit allocation or some arbitrary combination of both (further refer to as the *state* of the model), where the neural network creating the local exit network is taken as a black box.

Naturally, the best solution would be to find a closed form algorithm with model state as input and binary decision whether the local exit allocation has better accuracy then remote exit or not. Unfortunately, there is not known any algorithm (at the time of writing) capable of such decision in closed form a priori.

Another possibility is to come up with an algorithm capable creating the decision from the state with some error which is making such algorithm suboptimal solution, but in the case of small error an acceptable solution. This section has a goal to show some possible suboptimal solutions. Following three subsections describe three solutions further in text referred to as *uncertainty* or *uncertainty metric* for decision method offered in subsection 5.1, *MSBD* for decision method described in subsection 5.2 and *classifier* for decision method described in subsection 5.3

5.1 Model Uncertainty

An entropy is a confidence some classifier model has in its classification. This solution suggests similar approach with Bayesian confidence metric [20]. The solution uses random dropout during forward pass of local exit neural network to obtain variance of the allocation.

Dropout is a technique used for overfit reduction during training phase [21]. But if the same mechanism of random power reduction for the neural network is used during inference time it has an effect of introducing an allocation deviation the neural network would yield without it. Neural networks are still deterministic and passing the same input several times has always the same output. But, due to the random choice of dropout if dropout is applied in some arbitrary neural network and repeated predictions are made using this network, the predictions use slightly

different neural networks every forward pass and thus their results are not the same. Each of the predictions made by the neural network with applied dropout is deviated from the prediction the neural network gives without the dropout applied.

Definition 5.1 (Model Uncertainty Metric)**.** Lets have an arbitrary trained neural network model θ and sample S. If the sample is passed to the network it yields an allocation $\theta(S) = a$, a is a vector with length M. Lets each time a sample passes through the neural network a dropout is applied to the network with probability p. When sample S is passed to the neural network in $\mathcal{I}$ iterations the network gives $I = |\mathcal{I}|$ allocations a_i. Then

$$U(a_{\mathcal{I}}) = \left\| \frac{1}{I-1} \sum_{i \in \mathcal{I}} (a_i - \overline{a}_{\mathcal{I}})^2 \right\|_{\infty} \tag{19}$$

is said to be *uncertainty* of allocation a and mean $\overline{a}$.

The equation in the definition 5.1 has a bias caused by the mean value $\overline{a}_{\mathcal{I}} = \frac{1}{I} \sum_{i \in \mathcal{I}} a_i$ counted from finite number of allocations. There is a possibility to erase this bias by plugging allocation a produced by neural network without the dropout. This allocation is the mean from infinite number of allocations produced with the dropout probability p. Thus, uncertainty formula can be written as:

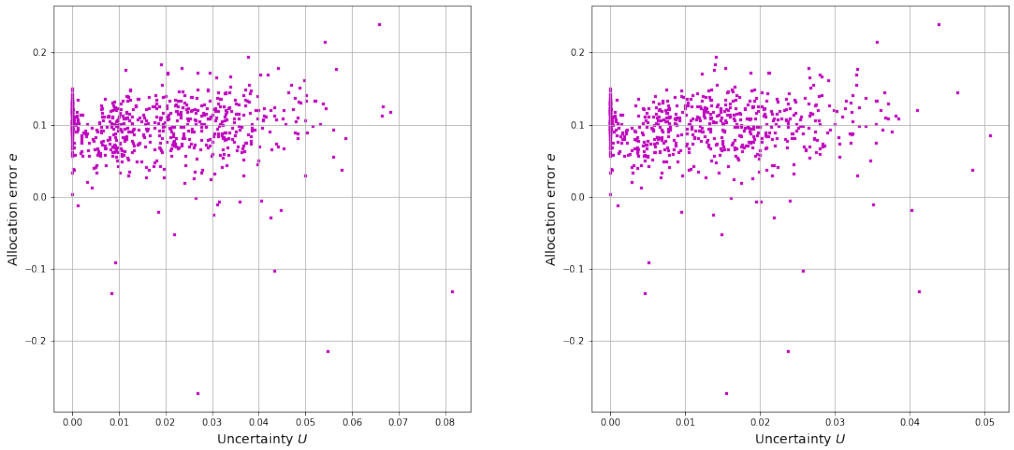
$$U(a_{\mathcal{I}}, a) = \left\| \frac{1}{I-1} \sum_{i \in \mathcal{I}} (a_i - a)^2 \right\|_{\infty}. \tag{20}$$

As depicted in Figure 11 uncertainty produced by either formula are not that different in distribution and using (20) does not change results in any significant way. Figure shows uncertainty counted by definition (19) is a bit shifted, but it is mainly caused by different realizations of allocations when uncertainties have been counted. But there is one positive in using (20) by reduction of redundant computation in form of allocation mean $\overline{a}_{\mathcal{I}}$.

The model uncertainty metric gives ability to measure Bayesian confidence, but it does not solve the decision problem. The model has to be set an uncertainty threshold which distinguishes between confident and non-confident allocations. Low uncertainty means confident allocation, thus, any allocation with uncertainty lower or equal to the threshold value leads to model yielding local exit allocation, otherwise model yields remote allocation.

Uncertainty metric based decision process has a disadvantage in additional search of the threshold value. Because, if the wrong threshold is set it can lead to lower overall performance by unnecessary increase of communication overhead or decrease of allocation accuracy. Also, uncertainty itself introduces an error, because it would ideally need infinite number of iterations, but for time saving purpose only $I = 10$ is performed in practice, which seems to be a good trade off between speed reduction and uncertainty accuracy.

Important part of the uncertainty metric measurement are which layer of the neural network should be dropout and with what probability. The most promising results are given when only last hidden FC layer in local exit is dropout and probability $p = 0.4$. Also an important observation have been made, the dropout should be used also in the training for uncertainty to yield results.



(a) Uncertainty counted by (19)

(b) Uncertainty counted by (20)

Figure 11: Comparison of difference in two uncertainty metric definitions shown at relation between allocation error e_t and uncertainty U for same set of samples and model trained with $w_l = 0.825$.

5.2 Consecutive samples similarity

Model uncertainty metric has some flaws, but one of them is also the speed reduction which has to be traded off with accuracy. Thus, there is a possibility to utilize a metric which would not have this problem.

The regression model takes into account a time window, which is the same for all input samples. Also, the model distribution separates samples which are easier to predict, that is why under powered local exit can predict them accurately, from samples which are hard to predict. What is easy and what is hard is problematic to say, but it can be seen that traffic similar to smooth functions (small noise) like sine for example can be very well predicted by under powered part of the model.

This observation can be utilized by looking at the sample and determine whether all time series are mostly smooth of not. Again, there is no clear way how to determine this. But there is one thing which can be observed about sample or its time series and that are spacial-temporal features. Signals with low noise can be recognised based on these features from those with high level of noise, because spatial-temporal features of two consecutive samples has to be alike or correlate a lot. The reason for it is that they are overlapping in majority of time steps. Thus, it can be defined how to measure correlation of two consecutive samples:

Definition 5.2 (Mean SBD)**.** Lets have two consecutive model samples $\boldsymbol{d}_{t-1,N}$ and $\boldsymbol{d}_{t,N}$ each containing same $\mathcal{M}$ time series and organized in the same way. Then Mean SBD is a metric counted as:

$$MSBD(d_{t-1,N}, d_{t,N}) = \frac{1}{M} \sum_{i \in \mathcal{M}} SBD(d^i_{t-1,N}, d^i_{t,N}). \tag{21}$$

This metric has exactly the same problem as model uncertainty with need of a threshold value, which has to be set prior of the model usage. But it has potentially an advantage in threshold value reusability since its value is connected to

performance capabilities of the model and not the samples themselves, meaning that potentially DDNN would be able to use the same threshold for decision process even after retraining for different local weight. But this last suggestion has to be a subject of the further research.

5.3 Machine learning based decision

Since there is no optimal algorithm known, which might be able to capture level of confidence of the model in its prediction, it can be possible and beneficial to use another machine learning supervised classifier, under some circumstances, for operating decision process in DDNN. Those circumstances are:

1. there is no easier solution or closed form solution,
2. distribution of such decisions is not changing over time and
3. possibility to train such classifier.

First condition is already satisfied, second must hold either way because the decision distribution can change only if states of the model will change, but in that case the whole DDNN model has to be retrained, thus this condition is not applicable. And at last the third condition will be described next.

The main goal of distributing the regression machine learning model over several nodes in the network is to limit communication over head between MEC and C-RAN (or other parts in the network depending where the DDNN is deployed). Thus, evaluation whether it is *worthy* to ask remote part of the DDNN for more accurate allocation has to take into account a possible penalty which sending some information over the network might add. Lets discuss how to evaluate whether and when the DDNN gains accuracy by choosing remote exit allocation as its output.

The easiest training of the classifier is supervised learning and in this case it can be easily done, because the model gives at any time two exit outputs for a single input which have also assigned a loss by objective function. In a sense objective function creates a metric helping decide which exit performs better over the input. Every input $d_{t,N}$ produces two outputs allocations $(\hat{y}^l_{t+1}, \hat{y}^c_{t+1})$. These allocations can be compared with true demand y_{t+1} and get pair of losses (l_l, l_c). Overall model performance is better in case of remote exit allocation when $(l_l - l_c) - c_p > 0$, where c_p is communication penalty which penalize remote exit for causing communication overhead. Otherwise model performs better by choosing local allocation. It can be seen as loss lower bound shifted by communication penalty.

Training itself needs a dataset which has to be gained from the already trained regression DDNN model. Lets assume already trained regression DDNN model with optimal local weight choice. It is trained with some dataset composing from samples $(d_{t,N}, y_{t+1})$. Then under assumption that the regression DDNN model is not overfit this data can be reused for training the decision classifier.

Dataset for training classifier consists of tuples (c_s, C), where c_s is a classifier sample and C is a decision class. Classifier sample consists of a vector with length $2M$. First M elements of the vector is a traffic input d_t and the second part of the vector consists of the local allocation vector $\hat{y}^l_{t+1}$. Classes C in terms of two DDNN exits are two and in dataset are represented as two tuples: $(1, 0)$ - return local exit allocation as the DDNN output and $(0, 1)$ - return remote exit allocation as the DDNN output.

Classifier structure is one hidden FC layer with 128 neurons followed by ReLU and output FC layer for two classes (local allocation, remote allocation).

Classifier optimisation objective is softmax cross entropy objective function [6]:

$$\mathcal{L}(\hat{v}^t, v^t) = -\frac{1}{|C|}\sum_{c\in C} v_c^t \log \hat{v}_c^t, \tag{22}$$

where $\hat{v}^t$ is probabilities of each class $c \in C$ and v^t is real classification probability of the input at time t. Also $\hat{v}^t$ is softmax value produced as:

$$\hat{v}^t = \frac{\exp \boldsymbol{z}}{\sum_{c\in C} \exp z_c}, \tag{23}$$

where $\boldsymbol{z}$ is a classifier output representing probability of each class $z_c, \forall c \in C$.

Training of the classifier is done by Adam optimizer with learning rate $lr = 0.0005$, which is the same as the learning rate for the DDNN itself.

5.4 Decision methods comparison

This section shows three possible solutions for DDNN decision process, in this part they are evaluated and compared to each other in one case scenario. All evaluations use dataset A subsection 4.4. DDNN model is trained using testing and validation parts of dataset A and set of weights from $\{0.1, 0.2, ..., 0.9\}$. Also all trained DDNNs are evaluated using test dataset of A. All trained DDNN use additive objective function in combination with constant under provision penalty (14).

After the training is done all DDNNs use algorithm 2, where as a decision process is plugged always one of the method previously mentioned in this section. Each decision has its own parameter, in case of uncertainty it is its *threshold* as well as it is the case for MSBD and classifier its training hyperparameter *communication penalty*. These hyperparameters influence performance of the DDNN as well as the training weights of the exits.

To evaluate how these hyperparameters affect DDNN performance they have to be compared with some theoretical value, which in this case is the loss lower bound. The loss lower bound is defined for optimal allocations at every time t and is fixed value of loss, but it also corresponds to some percentage of communication overhead O.

Communication overhead percentage O can be influenced by introducing again communication penalty, the exact same way as it is used in classifier dataset preparation. Then loss lower bound definition can be adjusted as follows:

Definition 5.3 (Penalized loss lower bound)**.** Lets have some DDNN trained for certain weight w_l using objective function f with some penalty p. At all time steps $t \in [0, T]$ DDNN creates two allocations, local allocation $\hat{y}_{t+1}^l$ and remote allocation $\hat{y}_{t+1}^c$ with respective errors e_t^l and e_t^c. Each allocation is assigned loss $\mathcal{L}_t^l = f(e_t^l)$, resp. $\mathcal{L}_t^c = f(e_t^c)$. Then *optimal penalized allocation* is said to be allocation chosen as:

$$y_{t+1} = \begin{cases} \hat{y}_{t+1}^c, & \mathcal{L}_t^l - \mathcal{L}_t^c - c_p > 0 \\ \hat{y}_{t+1}^l, & otherwise, \end{cases} \tag{24}$$

where $c_p \in \mathbb{R}^+ \cap \{0\}$ is communication penalty. Average loss for all $T + 1$ optimal penalized allocations is then called *penalized loss lower bound*.

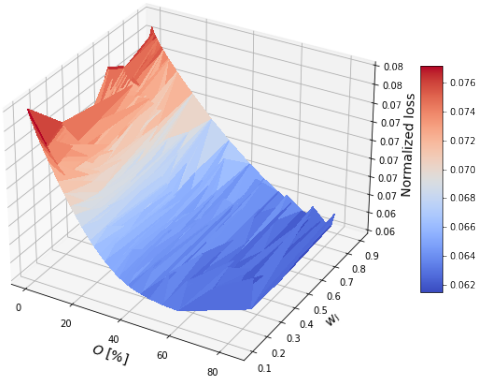


Figure 12: Normalized loss of DDNN as a function of communication overhead O and local weight w_l.

Definition 5.3 is giving possibility to found a relation between performance of DDNN respectively its average loss and the communication overhead. That adds additional hyperparameter to DDNN - communication penalty. Communication penalty has to be non-negative real value because there is no sense of negative values which would mean stimulation of communication overhead.

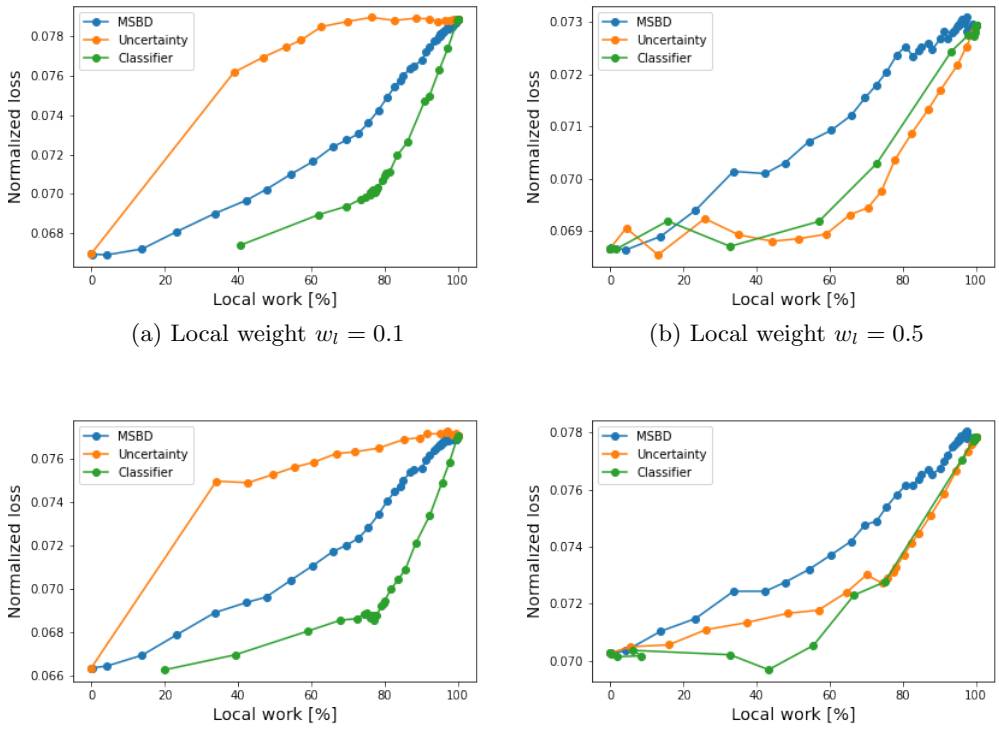
The DDNN loss is data dependent but since the distribution for samples during runtime should not change from the training dataset penalized loss lower bound function has to be relatively similar for all samples during runtime and can be seen at Figure 12. This function is produced by previously mentioned DDNN structure, dataset A, weight set and set of communication penalties $c_p \in \{0, 0.0025, ..., 0.15\}$. Also penalized loss lower bound is displayed normalized, where normalization is done by number of samples in dataset from penalized loss lower bound.

The important information about the DDNN in general is how well it performs in terms of its hyperparameters change, which is evaluated by objective function and also how this change communication overhead. The relation for the penalized loss lower bound was found by use of communication penalty. How is it for the decision process methods can be found as well. Each method has its *offload rule*. For example, the MSBD local allocation is chosen as final or overall allocation at the time t if the samples $\boldsymbol{d}_{t,N}$ MSBD value is lower than the threshold value. That means, function of loss can be found for decision process specific parameters.

In further text for better imagination communication overhead percentage is discussed about and displayed as *local work* and defined as complement of communication overhead percentage.

Offloading rule for MSBD, as already mentioned, is a condition whether the MSBD is lower or higher than the threshold. The same condition is applied for uncertainty decision as well. Finally, classifier trained for optimal penalized allocations is the last offloading rule.

Each method has different principle and thus normalized loss as function of method parameter does not have to correspond to each other. To ease the analysis a bit normalized loss functions for each method is displayed for different local weights w_l. Figure 13 shows different methods loss function comparison for four weights, where two of them are extreme cases of distribution.



(a) Local weight $w_l = 0.1$

(b) Local weight $w_l = 0.5$

(c) Local weight $w_l = 0.6$

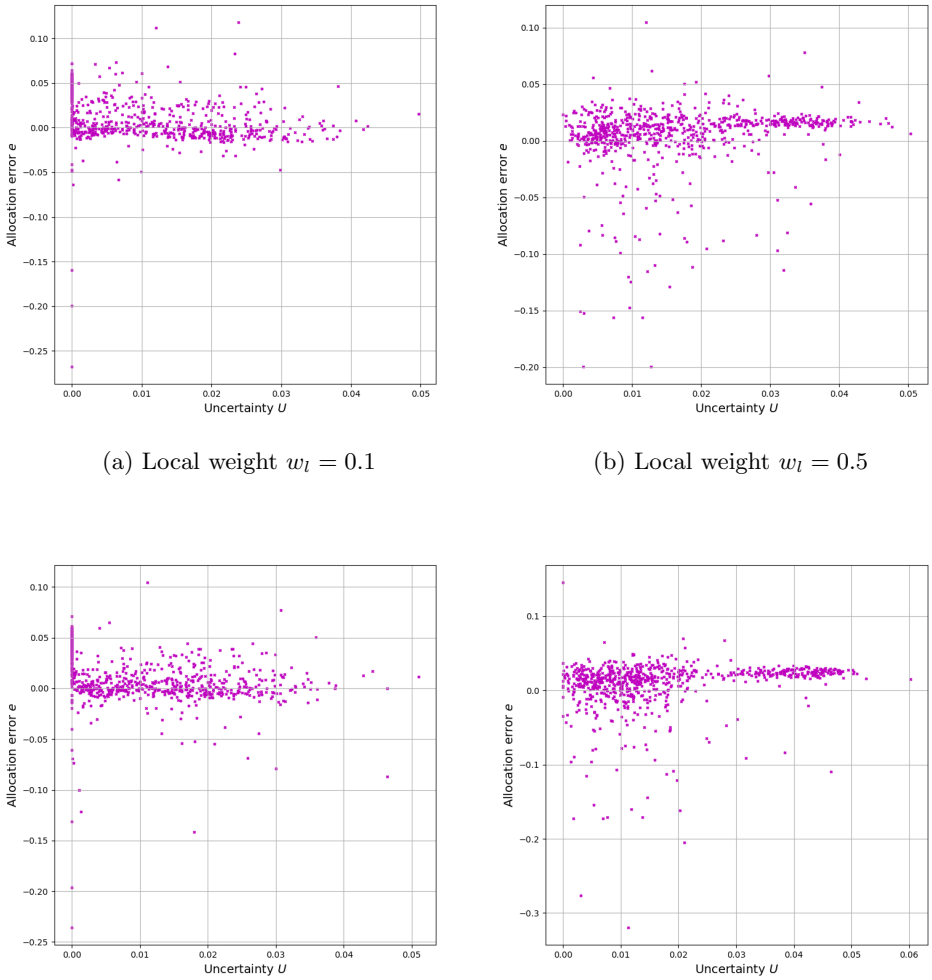
(d) Local weight $w_l = 0.9$

Figure 13: Decision process methods performance comparison for DDNN trained at multiple local weights. Plots show normalized loss as function of local work percentage with method specific offloading rule as hidden parameter.

The functions of normalized loss can be convex, concave, affine or neither of those for all local work percentages. Affine function means that sample loss increases with the local work and that any gain by distribution in communication overhead reduction would worsen allocations. Concave function means that if the level of distribution would be small the allocation would significantly deteriorate while for the big level of distribution the loss would be higher or nearly the same as the loss of purely locally made allocations. That is a behavior which have to be avoided and if any method shows this feature it should not be used. Concave function means that for high distribution level allocation loss is slightly higher or even lower then loss of fully centralized DNN. The last type of function which is neither of previous cases might lead to very erratic behavior, where slight change of local work percentage might lead to big changes in allocation loss.

From above function types the seek are convex functions. That can be seen also in Figure 12, where penalized loss lower bound is a convex function in all cases.

At the first look MSBD has nearly linear relation between normalized loss and local work. That is interesting because the meaning of this is that consecutive samples difference corresponds to sample loss and it seemingly does not matter on the local weight. That is exactly the meaning and the intention of the method. The decision is evaluated based on the sample difference from the previous one and is not influenced by the local exit performance in any way. If there is some profound trend other than the linear for this function that would mean that the DDNN local



(a) Local weight $w_l = 0.1$

(b) Local weight $w_l = 0.5$

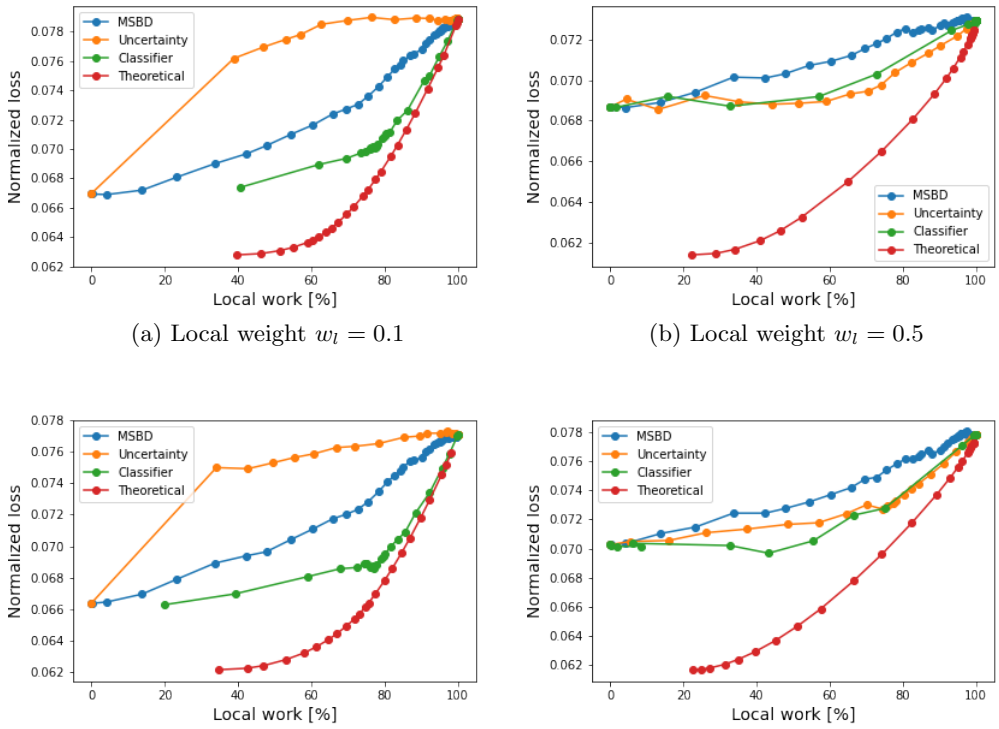
(c) Local weight $w_l = 0.6$

(d) Local weight $w_l = 0.9$

Figure 14: Model uncertainty metric as relation with local allocation error. Overview of multiple relations for DDNN trained with different local weights, showing changes in the relations.

exit reflects the dynamic changes in the samples as loss in the allocation. That is not happening leaving the conclusion that the DNN of local exit is not observing deviations in traffic samples. That can be also seen in Figure 5 even for centralized DNN, where DNN hardly allocates correctly immediate spikes in traffic, it more likely tries to allocate the sample based on its overall trend.

The next decision method, uncertainty metric, behaves erratically. For two out of four local weights normalized loss function is concave. This continuous plot is not very good for uncertainty metric, because is does not reflect qualitatively its behavior. Figure 14 offers relation between allocation error for each sample and its uncertainty. From two cases Figure 14a and Figure 14c can be seen that a some part of the allocations are assigned uncertainty equal to zero even though they have different allocation error. Thus, for the two cases in Figure 13 should be function discontinuous between first two data points. This work does not offer any



(a) Local weight $w_l = 0.1$

(b) Local weight $w_l = 0.5$

(c) Local weight $w_l = 0.6$

(d) Local weight $w_l = 0.9$

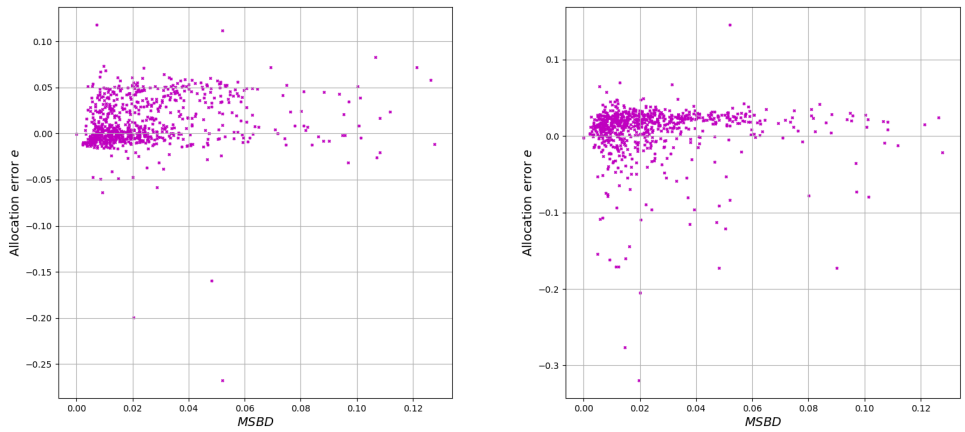
Figure 15: Decision process methods performance comparison for DDNN trained at multiple local weights. Plots show normalized loss as function of local work percentage with method specific offloading rule as hidden parameter as well as the penalized loss lower bound named as theoretical.

closer explanation of this phenomenon, because that would need more deep learning research in this area and the problem is more as an side track of the problem which this work aims to solve.

For purpose of the desirability of uncertainty can be just said that uncertainty based decision process can be used only for the local weights which does not experience this uncertainty behavior. Because for other two cases of weights where the normalized loss is continuous convex function it fulfils the expectation for the offload rule that up to certain point of distribution level or local work the DDNN allocations experience only slight allocation loss increase.

There is also another observation from Figure 14, there is no clear correlation between uncertainty and allocation error, which can be seen for all local weights. One value of uncertainty is assigned multiple values of allocation error. This fact undermines usability of the uncertainty as a reliable decision method. On the other hand in the Figure 13b and Figure 13d uncertainty clearly is able to create interesting function for normalized loss.

If the functions of normalized loss obtained using uncertainty might yield interesting results (slight loss allocation loss while significantly reduce communication over head) and also the correlation between uncertainty and allocation error is low there is an interesting question how would the normalized function look like whether the decision metric would be highly correlated with allocation error. The answer



(a) Local weight $w_l = 0.1$ (b) Local weight $w_l = 0.9$

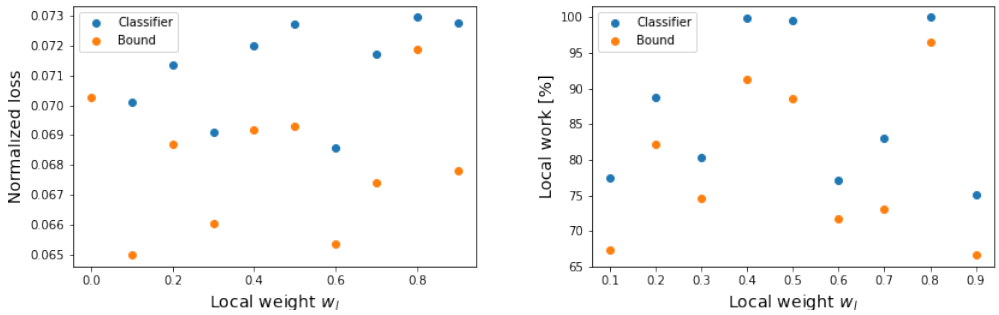
Figure 16: Comparison of two possible relations of MSBD and allocation errors for sample. MSBD has the same values for all samples in the both plots.

has to take into account the fact, that DNN distribution increases the overall DNN performance or better decreases the normalized loss of centralized DNN as Figure 6 shows. Loss lower bound already has some communication over head reduced and yet for all local weights the normalized loss is lower than the loss for centralized DNN. Taking this in a consideration then the potential of any metric fully correlated with allocation error has to be the penalized loss lower bound.

Figure 15 offers comparison of all decision methods and the penalized loss lower bound. Figures 15b and 15b shows how inefficient the uncertainty metric is in finding the theoretical bound. But not only the uncertainty metric is inefficient. There is also the last of suggested method classifier, which shows to perform the closest to the penalized loss lower bound. Uncertainty metric corresponds to classifier in two cases, which suggests both techniques might be equivalent for weights where uncertainty is able to evaluate all allocations with non-zero values. Classifier does not experience this problem and is able to perform well for all local weights, which is significant advantage over the uncertainty based decision.

Uncertainty is definitely more appealing version of decision then classifier due to classifier complexity. It takes a time to train the classifier, but this advantage is questionable during runtime where the classifier is faster especially for high number of iteration in uncertainty.

To compare all techniques with the penalized loss lower bound it can be seen in Figure 15 that only classifier is able to operate close to the theoretical bound. But it does so only for high percentage of the local work. There might be several possible reasons for classifier not to perform better for lower percentages, but most likely the classifier does not operate with enough information. For remainder, DDNN uses N passed traffic values for the data set $\boldsymbol{d}_{t,N}$ to create an allocation where classifier for its simplicity uses only the last instance of the traffic d_t and local exit allocation. Other possible explanation might be under powered classifier, 128 neurons in one hidden FC layer might not be sufficient to extract enough features to create quality decision.



(a) Normalized loss (b) Local work

Figure 17: DDNN with classifier performance in comparison with penalized loss lower bound for different local weights and communication penalty $c_p = 0.0225$.

For additional comparison between MSBD and uncertainty metric Figure 16 offers look at two relations between MSBD and local allocation error. As illustrated MSBD is not correlated much with local allocation error.

Choice of optimal decision process is two folded, at first choose a best performing decision method, which is the classifier and then choose hyperparameters for which the classifier yields allocations with the lowest normalized loss.

In practice network operator has to preset the communication penalty in advance. How to actually choose communication penalty is not a concern here but it can be for example scaled peak-to-average ratio of the network traffic without the overhead.

Once communication penalty is set, for example $c_p = 0.0225$, exhaustive search determines local weight w_l for which DDNN yields allocation with the lowest normalized loss. The grid search example for local weight compared with penalized loss lower bound is shown in Figure 17a. DDNN with lowest normalized loss is trained with local weight $w_l = 0.6$ even though the the lowest penalized loss lower bound is for weight $w_l = 0.1$.

Figure 17b shows respective local work percentages for DDNN trained with each local weight. It implies that DDNN is optimistic about its local exit allocation which yields higher local work percentage, but naturally worsens the allocation normalized loss.

For check how the DDNN trained for different local weights stands against centralized DNN Figure 17a shows value of normalized loss for centralized DNN marked as $w_l = 0.0$. It also shows that chosen DDNN model (DDNN trained at $w_l = 0.6$) is capable reduce communication overhead while also gain allocation accuracy (lower normalized loss per sample). Normalized allocation loss decreases by nearly 7% while local work is approximately 77%.

Theoretically local exit has to be able to create sufficient allocation at least for some fraction of samples, but if the local weight is penalizing the local exit performance to much it might not be able to create sufficient allocations. That is the intuition about the DDNN based on the (8). Figure 17 on the other hand shows that DDNN trained for the local weight $w_l = 0.1$ has the lowest penalized loss lower bound, with the classifier decision mechanism is able to still have normalized loss not worse than the centralized DNN while increase local work by comparable level as the chosen DDNN model. Also Figure 15 shows no signs of penalized loss lower bound deterioration for the local weight $w_l = 0.1$. Meaning of this is that the local exit

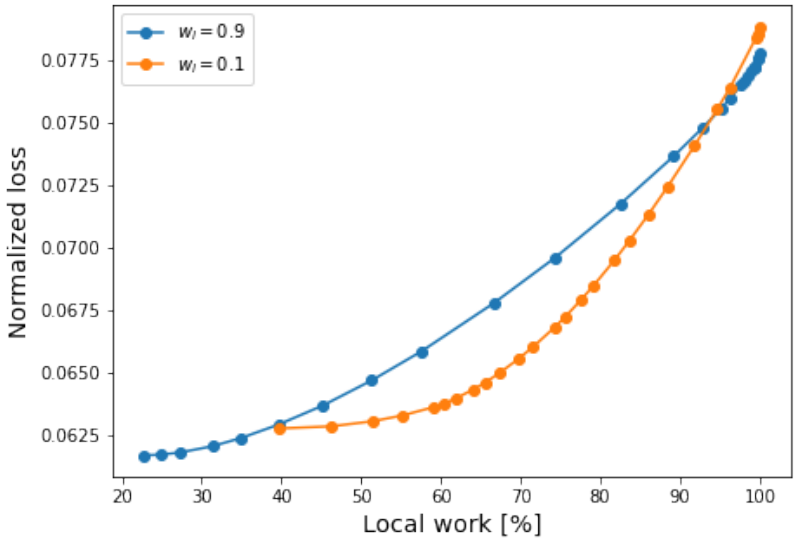


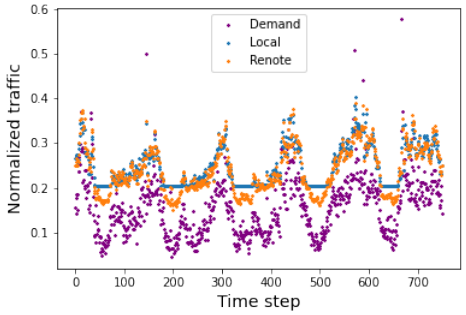
Figure 18: Penalized loss lower bound for two cases of local weights $w_l = 0.1$ and $w_l = 0.9$.

DNN importance might be substantially decreased, but the DDNN has potential to choose local allocation to moderate final allocation loss.

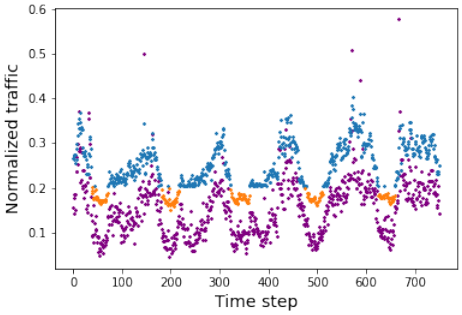
Figure 18 shows penalized loss lower bounds for weights $w_l = 0.1$ and $w_l = 0.9$. For the $w_l = 0.1$, the function even offers trade off with lower normalized loss for the same communication overhead.

To add more for decision methods comparison, each decision method yields final allocation for a data set, in which can be identified whether an allocation have been created by local or remote exit. Because each method has different principle they might not replace local exit allocations for remote exit allocations for the same samples. For purpose of comparison which samples or local allocations different methods chooses to replace for more accurate Figure 19 shows actual or also called true traffic demand for one BS taken as a VNF in the testing part of the set A. On top of BS traffic load Figure 19a shows all allocations created by local and remote exits. These allocations are made by the previously chosen DDNN trained for local weight $w_l = 0.6$. Local exit allocations suffers for poor accuracy in local minima of the traffic, which is caused by insufficient computational power of local exit trained with the high dropout $p = 40\%$. For closer observation the biggest allocation errors for local exit allocations are exactly in vicinity of BS traffic load minuma or spike changes of the traffic load trend. The optimal decision process should be able to recognize these two cases and replace local exit allocation with the remote exit allocation.

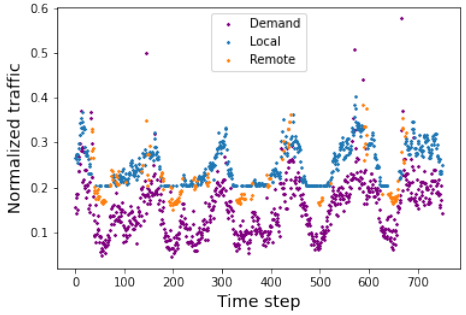
As shown in Figure 19b, classifier distinguishes local exit allocations in close to minima of the traffic load. These allocations have big allocation error and certainly remote allocation error of their respective remote allocations is lower. But the classifier fails to recognise and potentially improve sudden traffic spikes which are causing the under provision. That is a weakness of the classifier, it tries to improve mostly over provisioned allocations which is quantitatively right but not qualitatively. In this case the only possible solution for mitigation of high losses caused by traffic



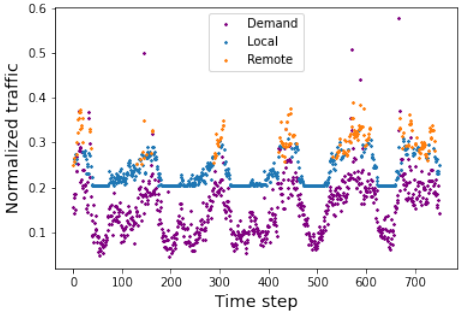
(a) Local and remote allocations comparison.



(b) Final allocations using classifier for decision.



(c) Final allocations using MSBD for decision.



(d) Final allocations using uncertainty for decision.

Figure 19: DDNN model trained for local weight $w_l = 0.6$ allocations with true traffic demand of one BS id 903. Comparison of local and remote exit allocations with each other and traffic demand. Overview of final allocations created with use of different decision methods.

load spikes is to making penalty policy stricter by increasing under provision cost p_c.

Increasing under provision cost has severe consequences in overall performance due to DNN preference to increase over provision, and as classifier rightly recognises increase in overall over provision error leads to worse performance.

The problem here is also a penalty policy function (14), which minimum shifts towards higher allocation error values if the under provision cost increases (penalty policy minimum is for allocation error $e = p_c\epsilon$).

In contrast with the classifier decision process is use of uncertainty metric for decision, which identifies peak values of the traffic load and yields remote exit allocation instead of the local one, Figure 19d. The identification of the values where the local (or even remote) DNN lacks accuracy is correct. The problem with this is that when a sample or one of its time series includes value off the traffic trend (lets focus only on sudden increase in traffic load) the DNN tents to mitigate it by increase over provision in the next allocation. The remote exit in these cases over provision the next allocation more than the local exit, Figure 19a. Thus, in this cases combination of the uncertainty and remote exit leads to higher error in the decision process.

The last Figure 19c shows behavior of the MSBD metric, which operates some-

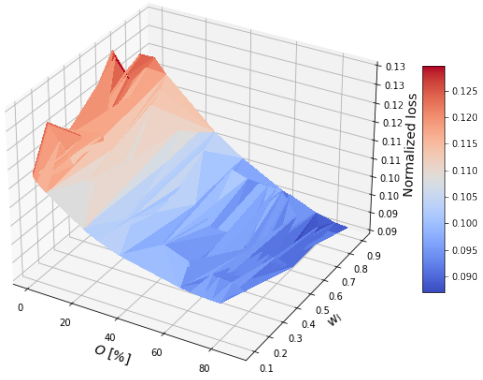


Figure 20: Normalized loss of DDNN as a function of communication overhead O and local weight w_l using coupling objective function and quadratic under provision penalty policy (13).

what in between the classifier and the uncertainty metric. As expected based on the principle of this decision method local exit allocations are replaced in final allocation by remote exit allocations when sudden change in the two consecutive sample occurs. The method is only traffic load data dependent, thus, for MSBD to perform worse that the other two methods because it lacks the knowledge of the allocation.

Figure 19 features only one BS out of the data set A. The allocations are created jointly as well as the decision about the whole local allocation $\hat{y}^l_{t+1}$. Thus some allocations $\hat{y}^{l,i}_{t+1}$ of one BS might be forced to use remote allocation due to the fact that the rest of the allocations worsen the error of the allocation $\hat{y}^l_{t+1}$.

The above mentioned comparison is for decision methods using DDNNs trained with the additive objective function. To demonstrate desirability also for the coupling objective Figure 20 illustrates penalized loss lower bound produced by DDNN trained for the same dataset A with coupling objective using quadratic under provision penalty (13), where constant $c = 50$ and slice pattern is *equal slices*.

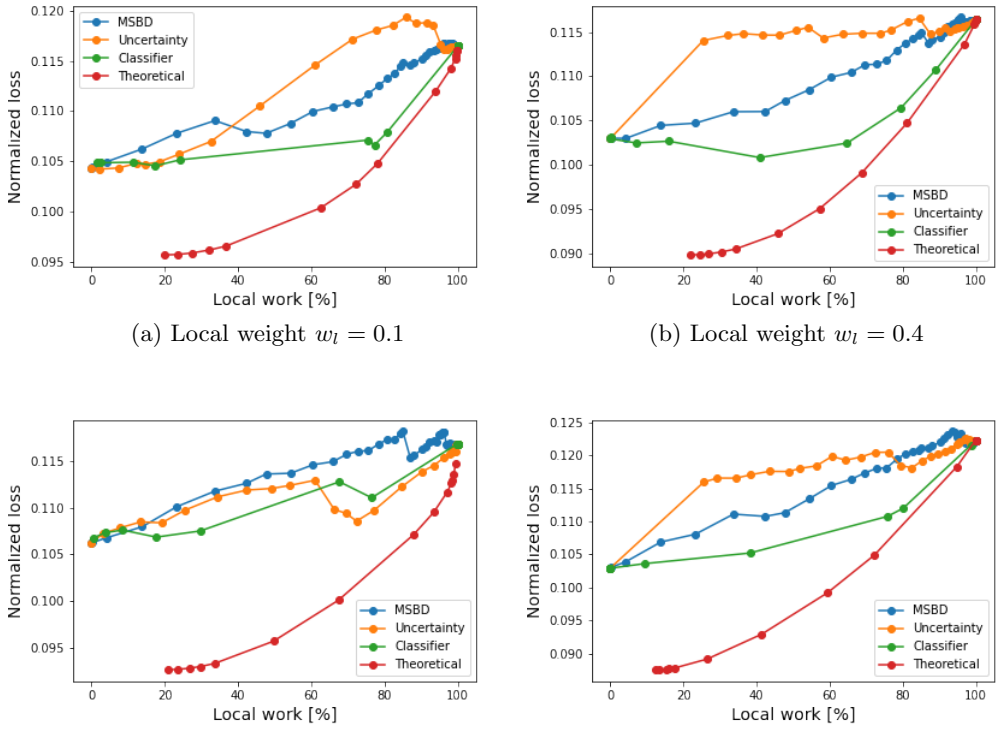
Coupling objective function using quadratic penalty policy also enables DDNN to find an operating point for local weight and communication penalty which benefits DDNN over centralized DNN in communication overhead while maintaining allocation capability. The question is how coupling objective influences each decision method.

By definition MSBD decision is not influenced by the objective function, thus its performance depends only on DDNN and its local weight w_l. Actual performance of MSBD decision method is shown in Figure 21. It follows similar performance for coupling objective function and additive one.

Figure 21 offers comparison of each decision method for a different set of local weights $w_l \in \{0.1, 0.4, 0.5, 0.9\}$. Local weight $w_l = 0.4$ is shown here instead of previously used $w_l = 0.6$ because it represents the best system of DDNN using coupling objective function.

For coupling objective function the uncertainty metric is not reliable in most cases it performs even worse than the MSBD. The reason why uncertainty metric is less efficient in terms of multiple slices would has to be a subject of further research.

From all decision methods classifier is the only one operating close to penalized



(a) Local weight $w_l = 0.1$

(b) Local weight $w_l = 0.4$

(c) Local weight $w_l = 0.5$

(d) Local weight $w_l = 0.9$

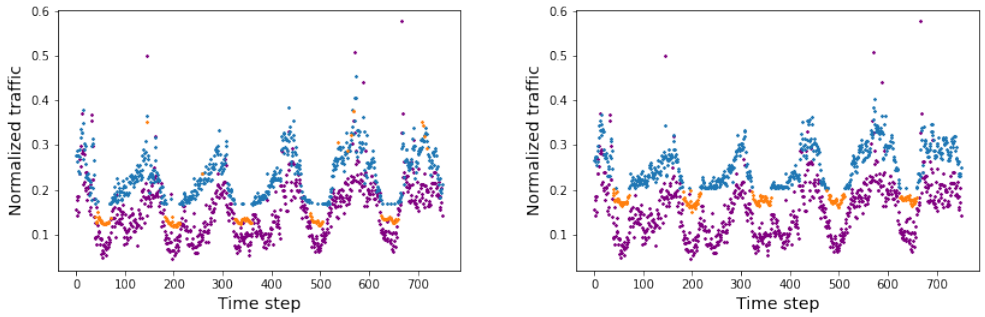
Figure 21: Decision process methods performance comparison for DDNN trained at multiple local weights. Plots show normalized loss, coupling objective function with quadratic under provision penalty policy, as function of local work percentage with method specific offloading rule as hidden parameter.

loss lower bound and performs similarly as in the case of additive objective function use.

How coupling objective influences final allocations can be seen in Figure 22 which shows comparison of previously discussed final allocations created by DDNN trained with additive objective function, Figure 22b, and final allocation created by DDNN trained with coupling objective function, Figure 22a. Both final allocations uses classifier for its decision process and chosen DDNN hyperparameters are chosen as those yielding the most accurate allocations for close percentage of local work.

Coupling objective forces VNFs allocations in one slice to be determined by the worst of them, thus, the main difference between final allocations in Figure 22 is final allocations yielded by the DDNN trained with coupling objective function have part of the allocations created by remote exit occurring in peak values of traffic demand. Displayed allocations are created for one base station and the influence of the rest of the slice in both cases can not be seen due to large number of base stations in the slice in both cases.

Final allocations produced with different objective functions seeks different goal as is written in the section 4 and can not be compared because of this fact in terms of them performance. But Figure 22 features also one interesting fact more, when allocations created by remote exit are presented in final allocation their error is smaller and can be seen by eye just from the figure. This fact is hardly caused



(a) DDNN trained with $w_l = 0.4$, $c_p = 0.04$ and coupling objective

(b) DDNN trained with $w_l = 0.6$, $c_p = 0.0225$ and additive objective

Figure 22: Overview of two final allocations for BS id 903 created using DDNN trained with local weight $w_l = 0.4$ and $w_l = 0.6$, communication penalty $c_p = 0.04$, $c_p = 0.0225$ and coupling objective function with quadratic penalty policy and equal slices and additive objective function with constant under provision penalty using data set A where decisions are made by classifier.

by the objective function more likely it is caused by use of different penalty policy. Penalty policy (13) using quadratic penalization of under provision has minimum of the function in $e_t = 0$ where (14) has minimum in $e_t = p_c\epsilon$. This shift forces DDNN to created over provision allocations.

From the decision methods comparison the classifier is the method which yields the most interesting results in terms of final allocation accuracy and communication overhead reduction. This fact is not influenced by the used objective function nor penalization policy. It is worth to develop such decision process even for its downsides in development and training time. The development would have to be redone or the offered classifier would have to be adjusted in case of any change in the problem like up scaling.

For possible extension any new decision method should be able to recognise where the remote exit allocation improves local exit under provision allocation to over provision. It should also be able to recognise traffic spike values and if the remote exit tends to push its allocation even further to over provision it should kepp the local exit allocation.

6 Conclusion

This work proposes a novel approach to the allocation of resources for network slicing with distributed DNN. The distribution separates layers of the DNN into two groups called local and remote. Local layers are located in C-RAN or Edge and remote layers are located in MEC or Cloud. Both types of layers can produce the same resource allocation differing only in accuracy of the allocation.

The DDNN is proposed to be trained for utilizing both exits in local as well as remote layers. The trained DDNN has an optimal operating point where certain percentage of local exit allocation are more accurate than allocations produced by the equivalent centralized DNN. The DDNN allocation is then combined from the local and remote exit. Whenever the local exit allocation is chosen to be a final DDNN allocation communication overhead between Edge and Cloud drops. The DDNN can be forced to yield increase percentage of local exit allocations in the final allocations to moderate communication overhead.

The DDNN performance increase and overhead moderation is then exploited to find an optimal trade off between the DDNN allocation accuracy and level of the communication overhead reduction.

The work on top this offers a new approach for slice allocation, where in contrast with previously offered solutions DDNN is capable jointly produce allocations for multiple slices by using coupling objective function at once.

Additionally, the work shows three different methods for DDNN performance moderation by allocation choice. These three methods are used as a priori machine learning regression accuracy comparators between local and remote exit allocations. The methods are compared in overall performance in single slice environment as well as multi-slice one.

The combination of all three parts creates an automated solution for resource allocation management capable of outperform centralized DNN while reducing communication, also for multiple slices.

Last, the work offers two suggestions for research extensions, where one suggests D-DRL as possible approach for version of the system for dynamic environment.

Any possible work extension can be made in two ares the work it self tries to extend the knowledge about. The first area is the further distribution or making the distribution more effective and the second area is to create an upgrade of the ML technique used for creating allocations.

To increase a level of distribution or to create more effective distribution technique would definitely have also impact in different fields than only telecommunications. As [6] suggested, distribution might be extended by using multiple local exit level parts for one remote exit. For example four C-RANs and one MEC would be able to accommodate DDNN model where four C-RANs playing a role of local exits. [6] shows a possibility to aggregate multiple local DNN output for MEC to be able to produce its classification. The aggregation is suggested by using max pooling, average pooling and concatenation. For regression purposes average or max pooling is not usable due to information loss. The concatenation is a problematic during runtime.

A multi local DNN distribution should work in three possible cases: (i) all of the model nodes decide to yield its local allocation, which work exactly like the local exit described in section 3, or (ii) all local nodes decide to yield remote allocation, then they their convolutional layer outputs has to be aggregated by the chosen rule

(concatenate) and the remote allocation would be yield including allocations for all input VNFs. The last (iii) possibility is then a fraction of local nodes decide to yield local allocation while the rest needs remote allocation. Then the problem of the concatenation occurs because the remote DNN would have not enough information for creating input and thus yielding its allocation. The suggestion for future is to effectively solve aggregation problem in this expanded system. Especially interesting question arises when one slice should be spread in multiple C-RANs and thus different parts of slice would be handled differently.

The next, change of machine learning technique can be a change from using DDNN towards utilizing DRL respectively D-DRL. A plus for reinforcement learning solution is a possibility to incorporate the decision process into its structure and based on the model experience at the time it would involve necessary parts for accurate allocation at each time step. Solution for this problem is not mere extension of this work because it needs to solve a few challenges like its internal structure for distribution, how to define reward mechanism for this structure able to reflect allocation accuracy, penalty policy and communication overhead level.

Any solution using D-DRL are interesting for its dynamic behavior able to adapt to changes in the environment and potential ability to reduce complexity in the system by omitting decision process entirely.

Reinforcement learning or any algorithm able to receive a feedback from the environment operating just for decision process might be an alternative to pure D-DRL solution. It might not be so appealing due to reducing the dynamic adaptability, but a form of reinforcement learning or solution using a form of multi armed bandits which would be able to utilize and extract information from multiple decision methods to get better performance might increase accuracy of the decision process. Such algorithm would greatly benefit especially when a context of the state would be offered.

References

[1] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys Tutorials*, 21(3):2224–2287, 2019.

[2] Jing Wang, Jian Tang, Zhiyuan Xu, Yanzhi Wang, Guoliang Xue, Xing Zhang, and Dejun Yang. Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017.

[3] Dario Bega, Marco Gramaglia, Marco Fiore, Albert Banchs, and Xavier Costa-Perez. Aztec: Anticipatory capacity allocation for zero-touch network slicing. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 794–803, 2020.

[4] Chaoyun Zhang and Paul Patras. Long-term mobile traffic forecasting using deep spatio-temporal neural networks, 2017.

[5] Dario Bega, Marco Gramaglia, Marco Fiore, Albert Banchs, and Xavier Costa-Perez. Deepcog: Cognitive network management in sliced 5g networks with deep learning. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, page nil, 4 2019.

[6] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Distributed deep neural networks over the cloud, the edge and end devices. *CoRR*, 2017.

[7] 5g; management and orchestration; concepts, use cases and requirements. *3GPP TS 28.530 version 16.2.0 Release 16*, 8 2020.

[8] Albert Banchs, Gustavo de Veciana, Vincenzo Sciancalepore, and Xavier Costa-Perez. Resource allocation for network slicing in mobile networks. *IEEE Access*, 8:214696–214706, 2020.

[9] Jose Jurandir Alves Esteves, Amina Boubendir, Fabrice Guillemin, and Pierre Sens. Drl-based slice placement under non-stationary conditions. In *2021 17th International Conference on Network and Service Management (CNSM)*, pages 225–233, 2021.

[10] Qiang Liu, Tao Han, and Ephraim Moges. Edgeslice: Slicing wireless edge computing network with decentralized deep reinforcement learning. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 234–244, 2020.

[11] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. *CoRR*, 2017.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978.

[14] John Paparrizos and Luis Gravano. k-shape: Efficient and accurate clustering of time series. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, page 1855–1870, 5 2015.

[15] Ingwer Borg and Patrick J.F. Groenen. *Modern multidimensional scaling : theory and applications*. Springer, New York, 2005.

[16] David F. Crouse. On implementing 2d rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–1696, 2016.

[17] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[18] Telecom Italia. Telecommunications - SMS, Call, Internet - MI, 2015.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[20] Lingxue Zhu and Nikolay Laptev. Deep and confident prediction for time series at uber. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, page nil, 11 2017.

[21] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, 2014.