# De-centralized Learning for Radio Network Key Performance Indicator Prediction

Jose Carlos Alcantara

Helsinki May 26, 2020

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

| Tiedekunta — Fakultet — Faculty | Koulutusohjelma — Studieprogram — Study Programme |
|---|---|
| Faculty of Science | Computer Science |

Tekijä — Författare — Author

Jose Carlos Alcantara

Työn nimi — Arbetets titel — Title

De-centralized Learning for Radio Network Key Performance Indicator Prediction

Ohjaajat — Handledare — Supervisors

Sasu Tarkoma and Antti Honkela

| Työn laji — Arbetets art — Level | Aika — Datum — Month and year | Sivumäärä — Sidoantal — Number of pages |
|---|---|---|
| | May 26, 2020 | 60 pages + 1 appendices |

Tiivistelmä — Referat — Abstract

A recent machine learning technique called *federated learning* (Konečný, McMahan, et. al., 2016) offers a new paradigm for distributed learning. It consists of performing machine learning on multiple edge devices and simultaneously optimizing a global model for all of them, without transmitting user data. The goal for this thesis was to prove the benefits of applying federated learning to forecasting telecom key performance indicator (KPI) values from radio network cells. After performing experiments with different data sources' aggregations and comparing against a centralized learning model, the results revealed that a federated model can shorten the training time for modelling new radio cells. Moreover, the amount of transferred data to a central server is minimized drastically while keeping equivalent performance to a traditional centralized model. These experiments were performed with multi-layer perceptron as model architecture after comparing its performance against LSTM. Both, input and output data were sequences of KPI values.

ACM Computing Classification System (CCS):
Computing methodologies → Machine learning → Learning paradigms

Avainsanat — Nyckelord — Keywords

machine learning, artificial neural networks, regression, time series forecasting, deep learning

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Thesis for the Algorithms, Data Analytics and machine learning subprogramme

# Contents

**1 Introduction** **1**

**2 Background** **3**

2.1 Time Series and Forecasting . . . 3
2.2 Deep Learning . . . 5
2.2.1 Exploding and Vanishing Gradients . . . 5
2.2.2 Data and Computing Availability . . . 6
2.2.3 Architectures . . . 7
2.3 Federated learning . . . 11
2.3.1 Federated averaging . . . 11
2.3.2 Comparison with centralized learning . . . 13
2.4 Transfer Learning . . . 15
2.5 Previous work . . . 16

**3 Methods** **19**

3.1 Data Description . . . 19
3.2 Train and test periods . . . 20
3.3 Data Sanitizing . . . 20
3.4 Autocorrelation . . . 22
3.5 Feature Scaling . . . 24
3.6 Clustering . . . 25
3.7 Embeddings . . . 28
3.8 Data Generators . . . 28
3.9 Learning Scenarios . . . 29
3.9.1 Multi-cell Scenarios . . . 29
3.9.2 Federated Learning Scenario . . . 30
3.9.3 FL Transfer Scenario . . . 30

**4 Models** 33

4.1 Baseline Model . . . . . 33

4.2 Metrics . . . . . 34

4.3 Optimizers . . . . . 35

4.4 Hyper-parameters . . . . . 37

4.5 Hyper-parameter Tuning . . . . . 38

**5 Results and Discussion** 39

5.1 Baseline Performance . . . . . 39

5.2 Model Selection and Hyper-parameter Tuning . . . . . 39

5.3 Embeddings . . . . . 41

5.4 Multi-Cell Learning . . . . . 42

5.5 Federated learning experiments . . . . . 46

5.5.1 Federated Updates . . . . . 46

5.5.2 Data Transfer . . . . . 46

5.5.3 Naive FL vs cluster-wise FL . . . . . 47

5.6 Transfer Learning Score . . . . . 48

**6 Conclusion** 52

**7 Future Work** 53

**Acknowledgements** 55

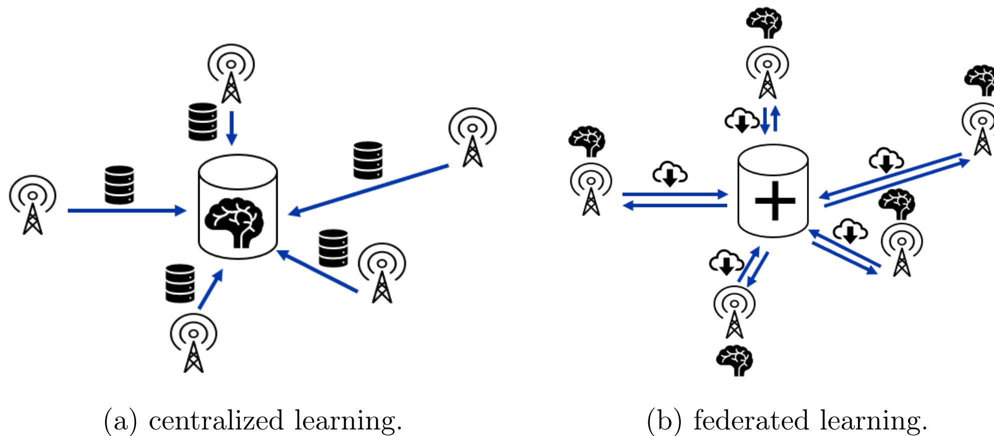
**References** 55

**Appendices**

**A Other Clustering Methods**

# 1 Introduction

Machine learning algorithms have traditionally been executed in one central location or entity (e.g. a computer, server or even a cluster) [MR17]. However, with the recent advancements in mobile devices' processors, Konečný, McMahan, et. al. from Google, developed a new machine learning technique called *federated learning* [KMY+16]. In contrast to *centralized learning*, in which data and processing take place at a central entity (e.g. a server), their main idea was to distribute data and its processing into a set or *federation* of these devices, coordinated only by a central server (Figure 1). As a consequence, this also had an impact in data privacy, since user data remains in-device and only model updates were transferred to a different location. However, the main goals for this thesis are evaluating the feasibility of distributed learning for radio networks and finding evidence of its advantages.



(a) centralized learning. (b) federated learning.

Figure 1: Centralized and federated learning. In centralized learning (Figure 1a) sampled data flows from devices to a central entity (e.g. server), which performs all of the training. In federated learning (Figure 1b) training is performed on-device and only updates are sent to the server for aggregation. Then, the updated global model is sent back to the devices.

To account for the advantages, two main hypotheses will be presented. The first one is that communication costs should reduce, given that the data is no longer transmitted to a central location (only the model weights) and that the computation is parallelized among the end devices. The second hypothesis is that training time should also diminish for a new radio network entity (in this case a radio cell) by transferring a trained federated model to it. This should give a better starting point than training this cell's model from scratch. If the previous hypotheses prove to be

true, I believe they could be generalized to other kinds of federations, for instance an Internet of Things (IoT) network for industrial control.

In order to prove these hypotheses, the federated learning algorithm was applied to a time series dataset provided by Nokia, which was generated from several radio network cells' data. The evaluation was performed by forecasting future values for different learning and training configurations, as well as for simulating the addition and modelling of new cells to a radio network. Forecasting of these values could be useful to allocate resources more effectively. For instance, if traffic were predicted, more bandwidth could be provided before a higher demand from users occurs.

Section 2 will introduce the necessary background from the fields of machine learning and time series. Next, the federated and transfer learning settings will be described. This section will also comment on similar distributed learning attempts. Section 3 will elaborate on the methods implemented for this research's experiments, such as how the data was initially analysed and pre-processed, clustering, embeddings, and learning configurations. Section 4 describes the machine learning models that were applied for performing the KPI prediction. It also depicts how these models were evaluated, optimized and tuned. Section 5 will present and discuss the experiments' results. Lastly, Sections 6 and 7 will summarize the findings from this thesis, leading to the concluding remarks and the suggested future work.

# 2 Background

In this section, the most relevant concepts and definitions will be briefly presented. Topics from time series data as well as machine learning will be covered.

Section 2.1 will introduce time series, given the nature of the dataset at hand. Next, deep learning will be characterized in Section 2.2. Having covered the previous, in Section 2.3 the federated learning scenario and its relationship to time series data will be described, particularly for radio networks. Section 2.4 will illustrate the technique called *transfer learning*, which was applied in this thesis's experiments (Section 5) to pass on a learned *federated model* to new cells. Lastly, Section 2.5 will comment about related previous work on time series modelling and distributed machine learning.

## 2.1 Time Series and Forecasting

Time series consist in ordered sequential data, distributed in regular time intervals [Die15, p. 234]. They are often represented as one or more random variables denoted by a time index. Some examples of time series are weather data, sensor measurements, product sales, and a country's population. The previous phenomena are known to change over time and one or more of their characteristic variables depend on previous information. Then, they can be modelled in order to get a forecast (also known as prediction). Predictions are calculated as a function of one or more time-dependent random variables and a set of parameters (commonly denoted as $\Theta$) [SS17, pp. 8-11,18,102-104]. An example of a linear model for forecasting can be observed in Equation 1, where $x_{t+1}$ (the next value of random scalar variable $x$ at time index $t+1$), is calculated as a function of its two previous values, $x_t$ and $x_{t-1}$, and adjusted by scalar parameters $\theta_0$, $\theta_1$ and $\theta_2$:

$$x_{t+1} = \theta_0 + \theta_1 x_t + \theta_2 x_{t-1} \tag{1}$$

where $\Theta = \{\theta_0, \theta_1, \theta_2\} \in \mathbb{R}$ and $x \in \mathbb{R}$ and $t \in \mathbb{Z}+$. Less or more previous values of $x$ with their corresponding parameters could be included depending on the desired model complexity.
Time series' main attributes are the following:

- *Trend.* It describes the movement of time series values' across long time periods (e.g. monthly trend, yearly trend). It can be increasing, decreasing or

constant[Die15, p. 235].

- *Cycles.* They are present in time series when a pattern repeats itself with a certain frequency, such harvest periods, product sales, or the moon cycle. We refer to seasonality when the cycles coincide with calendar factors such as holidays, weekdays/weekends and seasons [Nie19], [SS17, pp. 166.169].
- *Stationarity.* Stationary time series have constant mean, variance, and covariance. If a series is non-stationary, we can approximate the linear trend to some extent by differencing it or via linear regression [Pal16].

For time series the order of the data samples is relevant, in contrast to non-temporal data (like in image classification). Samples should also be continuous and learned in an incremental manner. This means that the sample rate should be uniform among the input samples and that the data should keep its order with respect to time (not shuffled). For instance, if there is daily sample rate, then the days from the input sample should be uniformly consecutive and uninterrupted. Otherwise, the relationship between samples will lose its significance and interpretation.

One notable advantage from learning the relationship(s) between time series samples is that the resulting model will allow forecasting of future values. Forecasting relies on the previous statistical properties (trend, cycles and stationarity). If the time series have cyclical values, patterns become clear and they become easier to characterize. It is also preferable that the time series are stationary. When time series are stationary, their variances are bounded and their estimated mean and variance remain closer to their true values. On the other hand, when time series are non-stationary, their variances are unbounded and tend to escalate over time, causing oftentimes that the forecasting model's prediction to under or overshoot [YM00, pp. 71-72].

A simple prediction example is the auto-regressive model, where a phenomena is represented as a linear relationship between variables [GWHT13, p. 61]. However, for this thesis, time series values had non-linear behaviour. Thus, they were fitted with non-linear models, more specifically, deep learning models (presented in the following section). The outputs were step predictions, whose number $n$ is a positive integer corresponding to how many steps (e.g. time indices or samples) ahead of the current one will the output value be [SS17, p. 102 -103].

## 2.2 Deep Learning

Deep learning is the product of recent advances in hardware and software, supporting machine learning algorithms. Success of deep learning occurred due to algorithmic improvements such as the idea of stacking multiple layers in artificial neural networks to achieve different levels of abstraction. For instance, in image classification, deep neural networks identify low-level features, such as edges, in their first layers; then shapes and lastly, high-level features such as faces or eyes. Deep learning's success was also possible due to the back-propagation [RHW86] and the gradient descent [Cau47] algorithms. Even nowadays they are one of the standard methods for training multi-layer neural networks [GBC16, pp. 18, 98].

### 2.2.1 Exploding and Vanishing Gradients

Further challenges for deep learning were the *vanishing* and *exploding* gradients. Both refer to the gradient update that is performed during back-propagation. When a neural network contains multiple hidden layers, the number of gradients to multiply increases as well. If the gradient values are large, then the updated weights can be moved too far away from their optimal value (a.k.a. *exploding gradients*). On the other hand, if gradients are too small, their update will become close to null as it propagates to the lower layers. This is called the *vanishing gradients* problem. The exploding gradient was tackled by a technique called *gradient clipping*, which consists in limiting the gradients' magnitude by a heuristic while keeping the direction of the update. The vanishing gradient was a more complex issue, which was counteracted with algorithmic techniques, such as batch normalization, weight initialization combined with ReLU activations, and the use of residual and LSTM networks [GBC16, pp. 175, 193-195, 290, 304, 317-321, 403, 415-416].

For the experiments in this thesis weight initialization was applied. The LSTM neural network architecture (See Section 2.2.3) was considered during model selection, and ReLUs for the candidate activation functions during hyper-parameter selection.

Rectified Linear Units (ReLUs) are to date one of the most common activation functions for deep neural networks. Examples of the most utilized activation functions are:

- **Linear:**

$$f(x) = x \tag{2}$$

- **Sigmoid:**

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

- **Hyperbolic Tangent (Tanh):**

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4}$$

- **Rectified Linear Unit (ReLU):**

$$f(x) = max(0, x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

- **Leaky ReLU:**

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.01 * x, & \text{otherwise} \end{cases} \tag{6}$$

- **Softmax:**

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \text{ where } i, j \in \mathbb{N} \tag{7}$$

One main reason for the popularity of ReLUs is that they are non-linear functions and when activated, their derivative is the same as for the linear function: the gradient will be 0 if the input value is less than or equal to 0; otherwise, it will be 1. This means that ReLUs can represent non-linear behaviour systems and that their derivative will not be affected by non-linear saturation, such as those found in the sigmoid and tanh activation functions. Furthermore, ReLUs can introduce sparsity into training, allowing for easier computation: only a subset of the neurons will have non-zero output/gradient. In addition, sparse activations have greater potential of being linearly separable [GBB11].

### 2.2.2 Data and Computing Availability

Another catalyst for the deep learning boom was the availability of larger datasets, which has allowed deep learning algorithms to reach human performance of tasks, most notably in the fields of computer vision and natural language processing [GBC16, pp. 18-21]. Together with this increment in data availability came also a considerable rise in computing load and processing time. This trade-off was tackled by the application of Graphics Processing Units (GPUs) combined with related

software developments. GPUs were noted to be fast and efficient for calculating matrix operations, which are commonplace when performing back-propagation and gradient descent in neural networks. Main software developments were the efficient application of distributed computing (including multi-GPU) as well as deep learning libraries, such as Theano, Tensorflow, Caffé, Keras and Torch.

### 2.2.3 Architectures

Deep neural networks can have different architectures, depending mainly on how the neurons are connected and activated. For this thesis, the architectures explored were Multi-Layer Perceptron (MLP) and Long Short-Term Memory (LSTM).

#### Multi-Layer Perceptron

The Multi-Layer Perceptron (MLP) consists of several layers of artificial neurons, connected as it can be observed from Figure 2. The first layer is called *input layer* and it is where the input data is provided for training and prediction. The intermediate layers are called *hidden layers* and within them, multiple neurons' interactions take place. If multiple hidden layers exist in an artificial neural network, then this process is called *deep learning*. Its interpretation is that at each deeper layer there is a higher level of abstraction from the learning objective. The last layer is called *output layer* and it is where the result is returned. In contrast to the single neuron or perceptron, MLP allows with its combination of neurons and activations, to solve more complex problems including non-linear classification and regression.

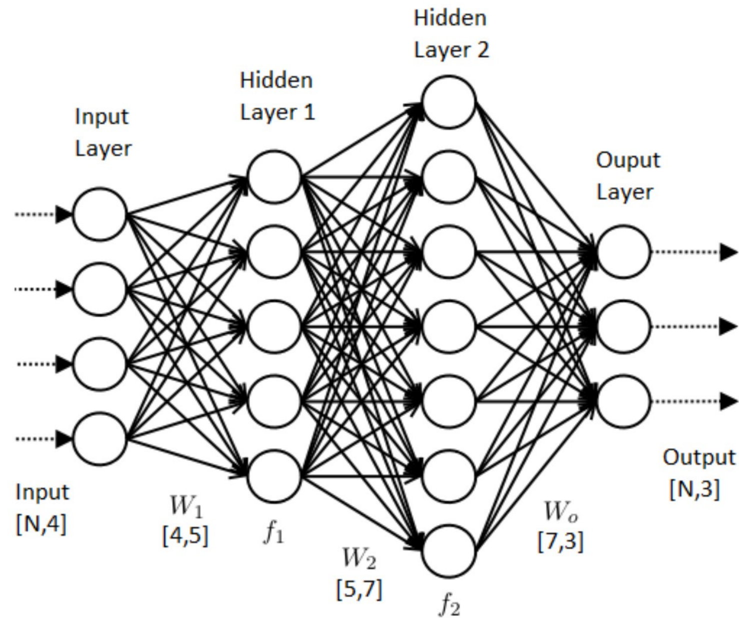



Figure 2: Example Multi-Layer Perceptron. A MLP is built by stacking layers of neural units (represented by nodes), where non-linear operations take place. Their connections are represented by arrows, indicating the direction of information flow. The basic layer types are: *input*, *hidden* and *output* layers. Image credits [via18]

**Long Short-Term Memory**

The Long Short-Term Memory (LSTM) architecture, developed by Hochreiter and Schmidhuber (1997) [HS97], is based on the recurrent neural networks (Rumelhart, 1986) [RHW86]. Recurrent neural networks (RNNs) were designed to deal with sequences, and examples of their main applications include text and speech processing. Each of their neurons contain a hidden state which depends on previous neurons' hidden states and/or outputs. In other words, the information flows from the past towards the present or future. In contrast to MLP, RNNs perform parameter sharing, meaning that the same parameters are used for different timesteps. Even though they process a sequence from step 1 to $\tau$, these step numbers are a relative position, not the actual values in time. In this manner, generalization for future sequences can be performed; otherwise, a set of parameters should be learned for each timestep to be introduced during testing/deployment [GBC16, pp. 373-374].

In spite of these advantages, RNNs suffer from gradient *vanishing* and *exploding*. An architectural solution for this issues in RNNs is the Long Short-Term Memory (LSTM). LSTM networks resemble RNNs, but with LSTM units instead of the typ-

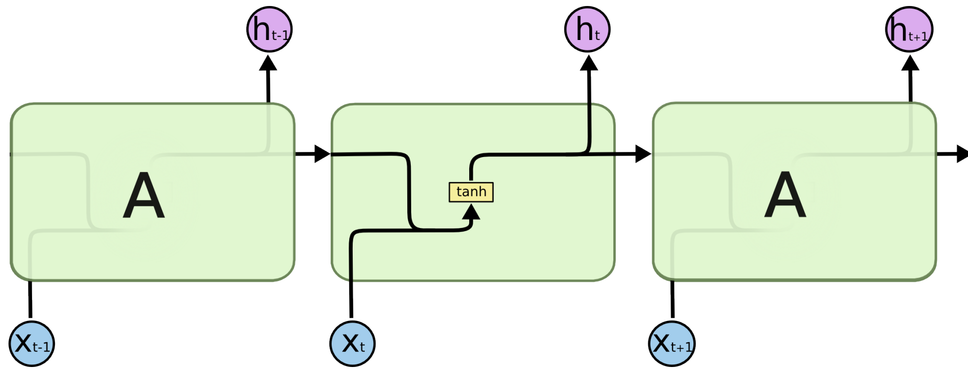



Figure 3: Unrolled RNN. Inputs at consecutive timesteps are represented by blue nodes $x_i$. Non-linear operations (e.g. tanh), occur at the recurrent neural units $A$ represented by rectangular blocks, producing hidden states $h$. The internal or hidden states are transmitted forward as input to the next RNN unit and they can be extracted from each RNN unit (as represented in the violet nodes) or only from the last RNN unit, to produce the desired output. Image credits [Ola15]

.

ical RNN hidden unit (with a non-linearity function after a matrix multiplication), as can be seen in Figure 4.

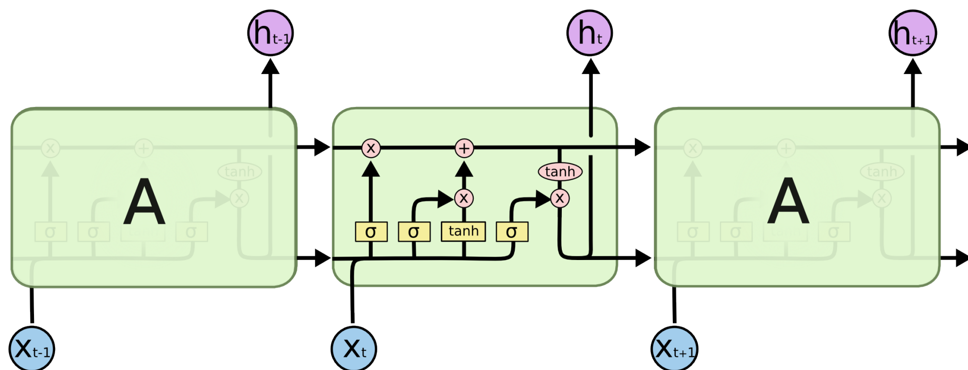
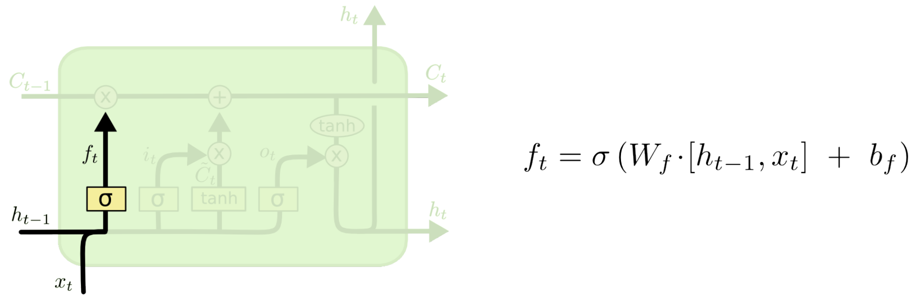



Figure 4: Unrolled LSTM chain. Notation for inputs and hidden states is the same as in Figure 3. However, for the LSTM units $A$, multiple operations corresponding to its *regulating gates* occur instead of RNN's single non-linearity. Hidden states travel forward through the lower arrows, and *candidate updates*, through the upper ones. Image credits [Ola15]

The LSTM unit is composed of *regulating gates* which control how much of the long-term information is retained and how much is forgotten. Their formulas are

simplified as follows:

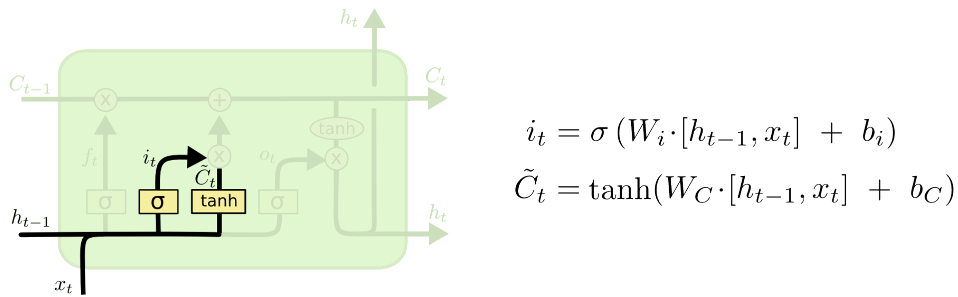
- ***forget* gate:** This section of the LSTM unit regulates with a sigmoid activation how much of the previous information will be forgotten.




$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 5: *forget* gate. Image credits [Ola15]

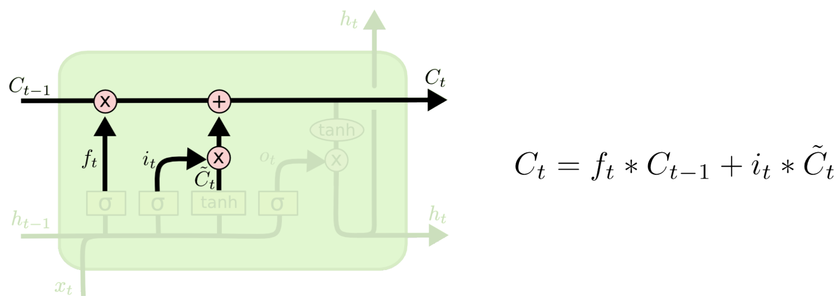
- ***input* gate:** This operation regulates how much of the input will pass to the next hidden state. $\tilde{C}$ is the candidate update.




$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 6: *input* gate. Image credits [Ola15]

- **update:** The update operation, being a combination of the candidate update and input gate value.




$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 7: update. Image credits [Ola15]

- ***output* gate:** It regulates how much of the hidden state will pass to the next LSTM unit or the final output.

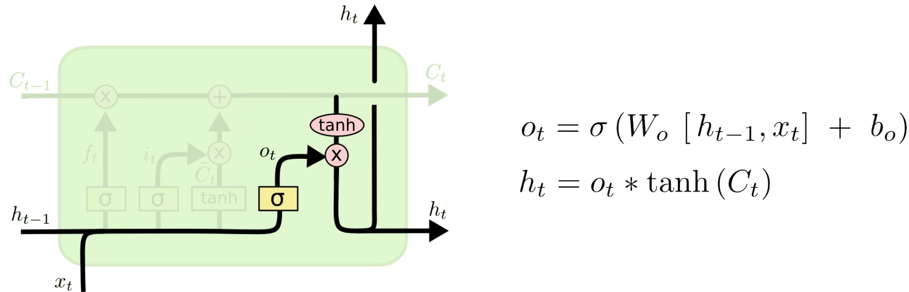



Figure 8: *output* gate. Image credits [Ola15]

With this configuration, then it is possible to control the information and memory flow of the neural network.

## 2.3 Federated learning

Federated learning (FL), as mentioned in Section 1, is a way of distributing machine learning. Having covered the required background, it can now be presented in detail. First, we should notice that the FL scenario contains two main types of entities: the global entity (e.g. server) and the local entities or clients (e.g. mobile devices). As a consequence, there will be global and local models respectively.

### 2.3.1 Federated averaging

The *federated averaging* algorithm (McMahan, et. al., 2016) [MMR$^+$16] is the core of federated learning (See Algorithm 1). Federated averaging (FA) begins by initializing the global model's parameters $w_0$ at the global entity or server (e.g. with random weight initialization). Then, a fraction $C$ of the clients $K$ are selected randomly (forming set $S_t$). Each of these clients receives the global model's parameters and performs one or more training rounds $E$ with batches $B$ of collected local data $P$. After training, each client sends back the updated parameters back to the server where they are aggregated as a weighted average. For this operation, each client's parameter's weight $w_{t+1}^k$ corresponds to the number of samples its client used for training $n_k$, divided by the total number of training samples from all clients $n$. The

results then become the global parameters and the cycle repeats for a limited number of rounds or indefinitely (depending on the application).

**Algorithm 1:** FederatedAveraging. The $K$ clients are indexed by $k$; B is the local minibatch size, $E$ is the number of local epochs, and $\eta$ is the learning rate. From [MMR+16]

**Server executes:**
    initialize $w_0$
    **for** *each round* $t = 1, 2, ...$ **do**
        $m \leftarrow max(C \cdot K, 1)$
        $S_t \leftarrow$(random set of $m$ clients)
        **for** *each client* **in parallel do**
            $w^k_{t+1} \leftarrow$ClientUpdate$(k, w_t)$
        **end**
        $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w^k_{t+1}$
    **end**

**Function** `ClientUpdate(`$k, w$`):` `// Run on client k`
    $\mathcal{B} \leftarrow$(split$\mathcal{P}_k$into batches of size $B$)
    **for** *each local epoch i from 1 to E* **do**
        **for** *batch* $b \in \mathcal{B}$ **do**
            $w \leftarrow w - \eta \Delta \ell(w; b)$
        **end**
        return $w$ to server
    **end**

Some highlights about the previous will be presented next:

- Federated learning requires the model to be parametric, which means that it should have a set of variables to optimize over training.
- Moreover, the model should be improvable in an iterative manner. In order to perform the averaging, all end-devices and the global model should have the same parameters.
- During federated averaging, the weighted average operation is performed with respect to the number of training samples belonging to each end device. Its purpose is to give higher relevance to updates obtained from more training samples, since statistically they represent better the current sample population of federated devices.

Typical data in a federated learning scenario comes from remotely connected devices. In contrast to the distributed learning approach, McMahhan, et. al. (2016) [MMR+16] describe this data by the following properties:

- **Non-IID** The training data for a local entity is generated by local sources (e.g. a user's mobile device or a telecom radio station). Thus, it will reflect particular traits of the user or geographical characteristics pertaining to the radio station, for example. Then, it can be concluded that it will be unlikely that it represents the whole federated population's distribution (or independent and identically distributed).
- **Unbalanced** Users, IoT devices and radio network stations are as well, unlikely to generate the same amount of data among their population. Thus, the training data as a whole will be unbalanced source-wise.
- **Massively distributed** Looking towards the future, it is expected that the number of federated devices will surpass the amount of samples per optimization round per source.
- **Limited communication** Remote devices are bound to have varying latency and connectivity. Thus, data samples will not flow at a constant rate. Moreover, devices will not always be online.

### 2.3.2 Comparison with centralized learning

Even though a central entity remains involved in federated learning, it still has significant differences with respect to centralized learning. One of them is that all of the user data remains in the mobile devices, and it is processed in each of them as well. Furthermore, only the learned parameters and number of training samples are sent back to the server. As a consequence, less data has to be transferred, reducing latency and bandwidth. For telecom companies these aspects could translate into monetary savings. Furthermore, this configuration introduces an extra layer of privacy for the end-users, since their personal data never leaves their device [BEG+19].

A shortcoming from federated learning, in comparison to the traditional approach (e.g. with supercomputers as central servers), could be the lack of high-end processors and GPUs which can enable faster updates. This would be expensive to have in thousands of mobile devices. However, at some point, distributing the processing at a large enough amount of devices can surpass this bottleneck. Considering that

there are globally more than 3 billion smartphones [New18] and 7 billion connected devices [Las18], and that AI chips are now present in phones from major companies, such as Samsung, Apple, Huawei, as well as in Amazon's IoT interfaces (Echo and Alexa) [Ago18], I am convinced that if not already, in the upcoming years, federated learning devices will surpass the capability of high-end servers.

Another difference with respect to centralized learning is that when modelling data from mobile devices, not all of them might be connected at the same time and communication latency will also vary among them. Centralized algorithms are not designed for this scenario; nonetheless, federated learning does, by updating the global model with samples of connected devices, and then updating the whole federation as soon as the devices meet the right conditions (e.g. battery, bandwidth) [BEG+19].

Considering the case for this thesis, by installing lower or mid-range GPUs at base stations, the relative cost would not be as high as installing them in mobile phones. In addition, base stations are more likely to have a stable and continuous connection than other mobile devices (e.g. IoT gadgets and phones), which had been identified as a weakness of FL [KMY+16].

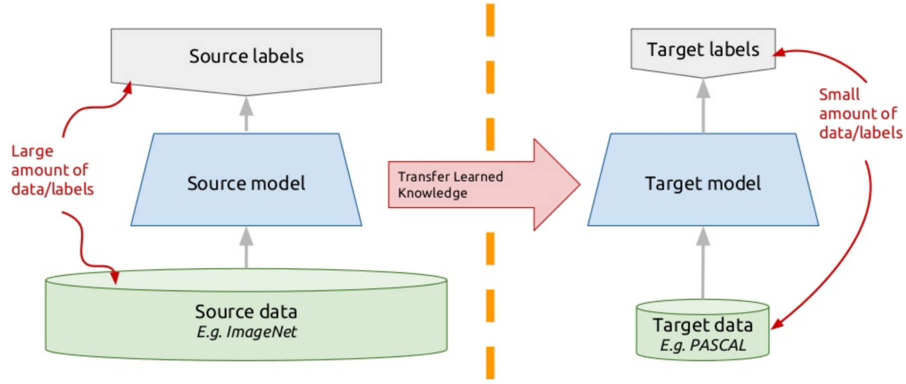
Given these advantages, I trust that federated learning has potential for providing a distributed, cost-effective, and more robust solution for telecom.

## 2.4 Transfer Learning

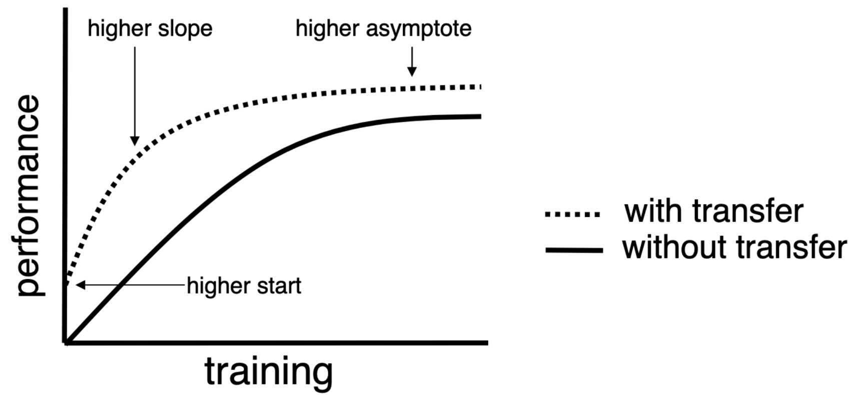
As mentioned in Section 1, another experiment of interest is transferring a federated model to a new cell within a telecom network. Transfer learning basically consists of taking advantage from a previous dataset or model in order to attain two main goals:

- To have enough data for training an algorithm when data is scarce. Adding data from a similar domain or application to augment the dataset, or transferring a model trained on similar data can compensate for its absence. Furthermore, it can help to generalize given shared features [GBC16, pp. 536-541] (See Figure 9a).
- To accelerate the learning process of an algorithm. When a trained model from a similar application and/or domain is available it can be used as a better starting point for learning, instead of doing so from scratch [GBC16, pp. 536-541] (See Figure 9b).

For the current case, a previous model was obtained by training a federation of radio cells over the prediction of a target KPI. Then, this model was transferred to a new cell whose data was kept aside, so that both its score and learning curve could be compared with respect to training its own model from scratch.




(a) Scarce Data. Image credits [McG17]




(b) Training Acceleration. Image credits [TS10]

Figure 9: Transfer Learning Advantages. The diagram in Figure 9a represents the knowledge transfer between a previous (source) model and a new (target) model. Figure 9b shows the expected learning curve resulting from transfer learning.

## 2.5 Previous work

In contrast to recent deep learning neural network models, classical time series forecasting has been addressed by statistical autoregression models (AR). These are commonly combined with moving-average models (MA), resulting in hybrid models such as ARMA (Whittle, 1951) [Whi51] and ARIMA (Box and Jenkins, 1970) [BJ70]. With this combination, they can account for the current value of the target variable, as well as for its random variation, given previous observations. AR, MA and ARMA require that the data is stationary. In contrast, ARIMA removes most of the non-stationarities as part of its algorithm by differencing the random variable or input $x_t$. ARIMA also performs model selection (AR or MA) by evaluating AIC

and BIC information criteria [Zha18].

An autoregressive model is a function of parameters $\phi$ and inputs $x_{t-1:p}$ and Gaussian noise $w_t$, where $p$ is the order. An autoregressive model of order $p$ or AR(p) is expressed as:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + ... + \phi_p x_{t-p} + w_t \tag{8}$$

A moving average model approximates the noise (or errors from past forecasts) $w_t$ linearly with respect to parameters $\theta$. A moving average model of order $q$ or MA(q) is expressed as:

$$x_t = w_t + \theta_1 w_{t-1} + \theta_2 w_{t-2} + ... + +\theta_q w_{t-q} \tag{9}$$

Thus, an ARMA model of orders p and q or ARMA(p,q) has the form:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + ... + \phi_p x_{t-p} + w_t + \theta_1 w_{t-1} + \theta_2 w_{t-2} + ... + +\theta_q w_{t-q} \tag{10}$$

If $p$ were 0 and q were greater than 0, then it would be equivalent to the moving average model; in the opposite way, it would be equivalent to the autoregressive model [SS17].

In addition to ARMA's autoregressive order $p$ and moving average order $q$, ARIMA adds parameter $d$, which stands for the order of differencing for $x_t$.

More recent approaches to solve the time series analysis include linear machine learning methods (e.g. linear regression, lasso), non-linear machine learning methods (e.g. random forests, support-vector machines) and ensemble methods (e.g. random forests, boosting algorithms).

The closest distributed machine learning algorithms to federated learning, according to [BEG+19] and [MMR+16], are the ones presented in *Large scale distributed deep networks* (Dean, 2012) [DCM+12], *Distributed GraphLab* (Low, Y., Bickson, D., et. al., 2012) [LBG+12], *Distributed training strategies for the structured perceptron* (McDonald, 2010) [MHM10], and *Parallel training of deep neural networks with natural gradient and parameter averaging* (Povey, D, Zhang, X. and Khudanpur, S., 2014) [PZK14]. However, "these algorithms do not consider datasets that are unbalanced and non-IID" (McMahan, et. al., 2017) [MMR+16]. Federated learning, by design, considers the data that these approaches fail to support (See Section 2.3.1) and that "each client will only participate in a small number of update rounds per day" [MMR+16].

For the case presented in this thesis, radio station datasets are likely to be unbalanced and non-IID, as users' behaviour changes over time or geography. For instance, there will not be the same amount of connected users in daytime hours as in night-time hours, or during week days as during weekends. In a similar way, data models learned from radio stations in the countryside are not likely to be enough to generalize for behaviour in cities. However, it should be noticed that radio stations are less prone to be as massively distributed as mobile phones.

# 3 Methods

In this chapter the procedures for data selection, preprocessing, and batch generation will be described. Sections 3.1 to 3.5 will introduce the telecom dataset and how it was pre-processed, with procedures such as splitting it (into training, validation and test data), removing invalid samples, handling missing samples, and scaling the different feature values. The rest of the chapter will be about how different models were trained during my experiments. Sections 3.6 and 3.7 will address alternatives for capturing similarities within the data sources and for encoding timestamp data. Section 3.8 will provide details about how data was transformed into training, validation and test batches to train the deep learning models. Finally, Section 3.9 will present different scenarios that were explored for learning the telecom data.

## 3.1 Data Description

The data was collected from 33 radio cells for over 4 months at 15 minutes intervals. Each record contains the radio cell's id, the timestamp and values from 180 different Key Performance Indicators (KPIs). See Table 1.

Table 1: Time Series Dataset

| **Cell ID** | **Timestamp** | **KPI 1** | **KPI 2** | ... | **KPI 179** | **KPI 180** |
|---|---|---|---|---|---|---|
| Number or hashed ID | Year-Month-Day Hour:Minute:Second | Numeric value | Numeric Value | ... | Numeric Value | Numeric Value |
| ... | ... | ... | ... | ... | ... | ... |

From the 180 available KPIs, Nokia domain experts selected 11 from the business perspective and their causality. The target KPI was then further selected (from these 11) by observing the auto-correlation plots (see Section 3.4). Thus, these 11 KPIs (including the target KPI) would be used for predicting the next value(s) of the target KPI.

## 3.2 Train and test periods

The data was not collected continuously as it was revealed by its time plots (See Figure 10), which is not optimal for time series prediction as discussed in Section 2.1. Thus, the longest semi-continuous period was chosen for performing experiments. This period contained records from August 6, 2018 till September 17, 2018 making a total of 6 weeks or 42 days. Then, the first 5 weeks were selected as the training set and the last one as the testing set for the experiments. Weeks were selected as measurement for two reasons: because of the shortness of the dataset and because of user's behavior cycles (again confirmed by Nokia's domain experts); for instance, weekdays' and weekends' behaviors.

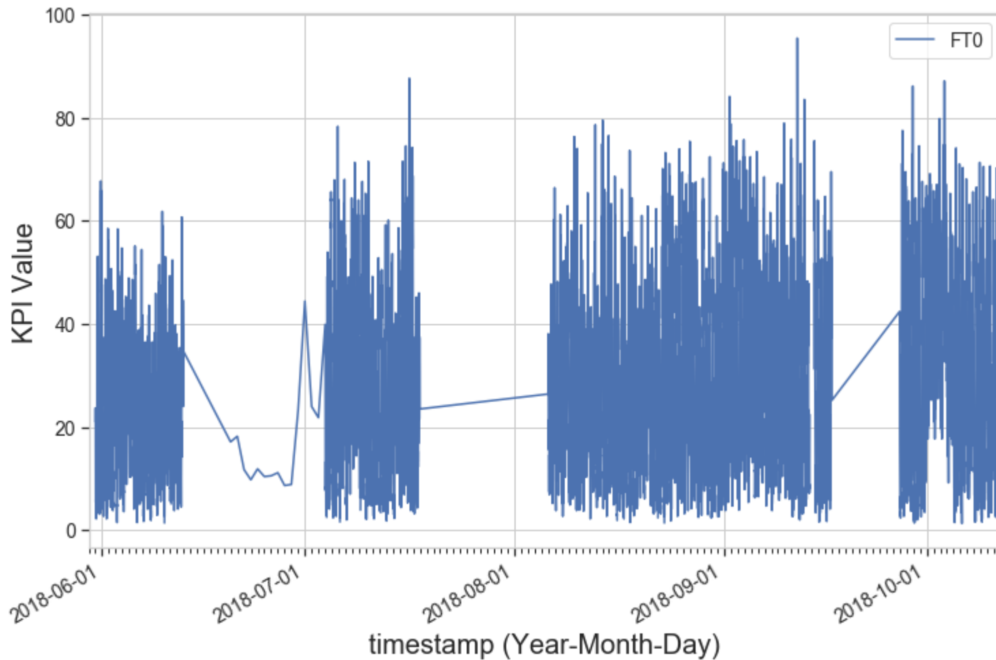



Figure 10: Target KPI Time Series. This plot is helpful for understanding the target variable in terms of its magnitude range as well as its sampling periods. By taking a closer look as in Figure 12 we will be able to observe more properties.

## 3.3 Data Sanitizing

After selecting the dataset by its relevant KPIs and by its time period, the following measures were taken in order to have clean train and test data. Although the number of missing values per feature were relatively small with respect to the total number

of records (see Table 2), their proportion within each radio cell could be significantly larger. Therefore, missing values (or NaNs) were accounted for by dropping radio cells with missing value percentage higher than 10% for any feature within the train or test periods. This resulted in 25 radio cells fulfilling the condition. The purpose for this was to interpolate the missing values afterwards with as small as possible amount of missing values per feature and per radio cell, so that most of the real data would remain in the datasets. In addition, wider gaps that could not be interpolated were filled with the mean value of the series. There were also some records sampled out of the 15 minute sampling rate, which were dropped in order to keep consistency of the time series.

Table 2: Missing values per feature. Total number of records: 224025

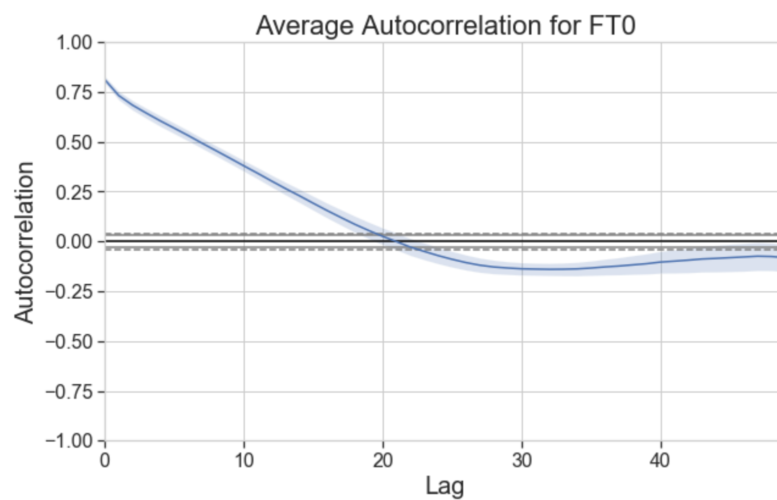
| **Feature** | **Missing Values** |
|---|---|
| ID | 0 |
| Period start time | 0 |
| FT0 | 1411 |
| FT1 | 992 |
| FT2 | 1531 |
| FT3 | 1531 |
| FT4 | 992 |
| FT5 | 1529 |
| FT6 | 1531 |
| FT7 | 1411 |
| FT8 | 1531 |
| FT9 | 1543 |
| FT10 | 1411 |

## 3.4 Autocorrelation

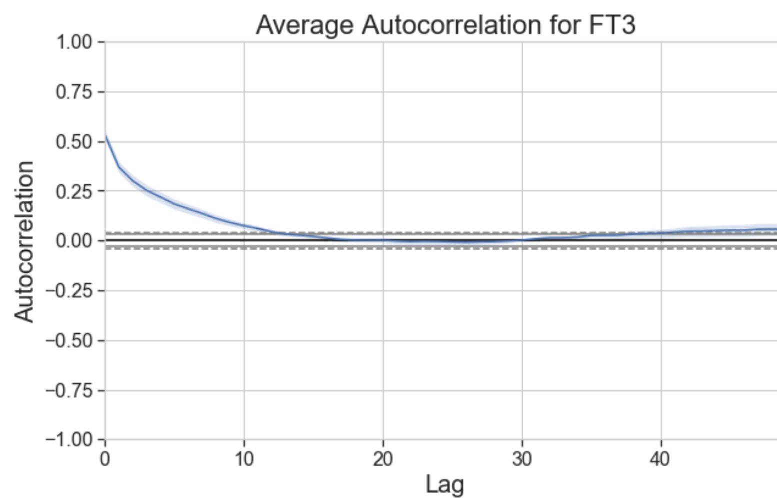
With sanitized and uniformly sampled data within a consistent period, the target KPI could now be selected. As mentioned in Section 3.1, it was chosen after applying the autocorrelation function (ACF) to each of the 11 pre-selected features. Autocorrelation is a measurement of how much related (correlation) is the present value from a time series with respect to its previous values (lags) [GWHT13, p. 94]. In order to know how much steps ahead can we forecast given time series data, we should observe its autocorrelation plot, which is the representation of the ACF. Its resulting values range from (-1,1):

- **1** being 100% correlated
- **-1** being 100% inversely correlated
- **0** being 0% correlated

In Figure 11 the autocorrelation plot can be observed for the selected target feature, which has high autocorrelation, as well as for a lowly autocorrelated feature for comparison.




(a) High autocorrelation example. The target variable FT0 has high autocorrelation as its ACF plot displays values above 0.5




(b) Low autocorrelation example. FT3 has low autocorrelation as its ACF plot only displays a value above 0.5 for the 0 lag setting.

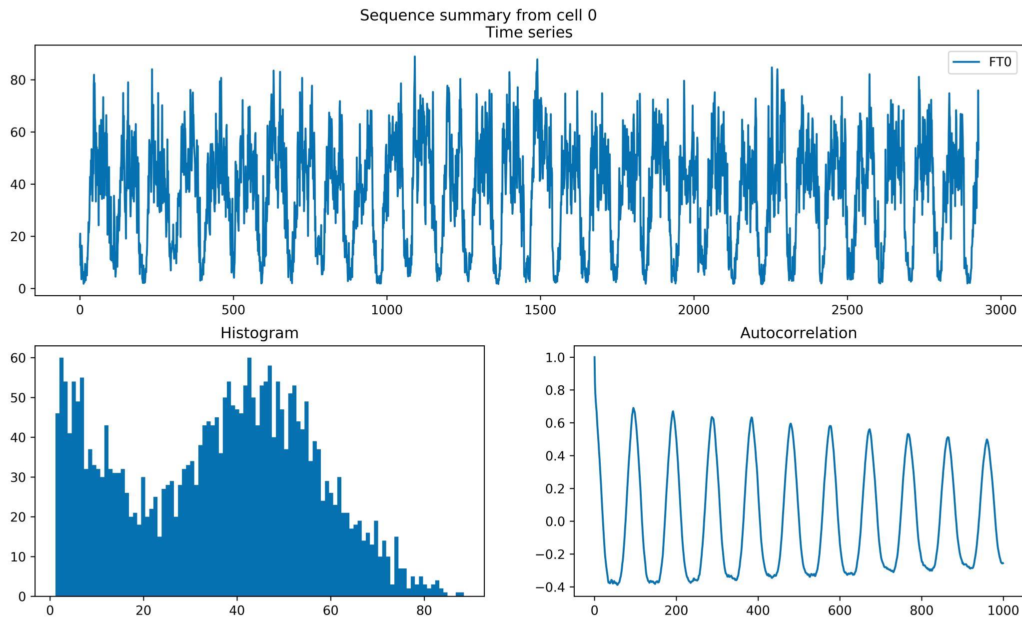
Figure 11: Autocorrelation Examples

## 3.5 Feature Scaling

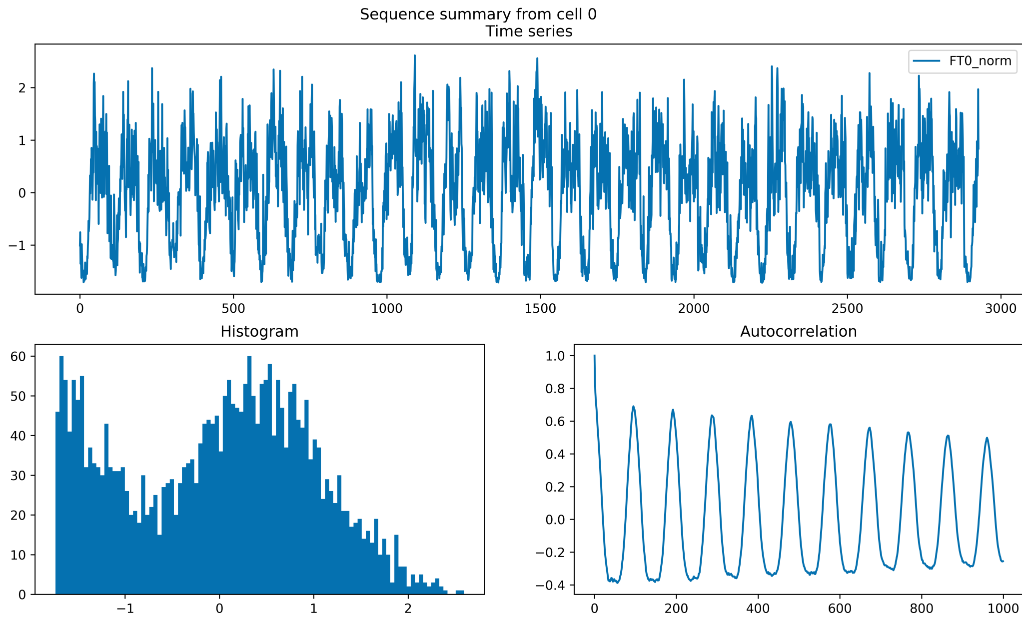
Since values among features varied in terms of magnitude and distribution, the input data (independent variables) was standardized. This was performed by removing the mean and scaling the data to unit variance. If this did not happen, there might be unbalanced feature contribution, making the algorithm's optimizer to learn unevenly as well. This process was applied feature-wise in the training set and the obtained scaling parameters were applied to the validation and test sets. Moreover, it was verified that the pattern, distribution and auto-correlation were preserved after scaling (See Figure 12).

MinMax scaling was also tried. This method scales the vector's values into predefined minimum and maximum values, in this case 0 and 1. However, the results were not as satisfactory as with the previous method (Standard Scaling).

For both, the default parameters of StandardScaler and MinMaxScaler from the sklearn library were used.
At this point data preprocessing has been covered. The methods in the following sections will describe how the models were trained.




(a) Cell0: Time series visualization of target KPI without scaling




(b) Cell0: Time series visualization of target KPI with scaling

Figure 12: Scaling Example: Cell0 PRB time series before and after scaling. Produced with code provided by Nokia's MN SoC R&D ML&AI team

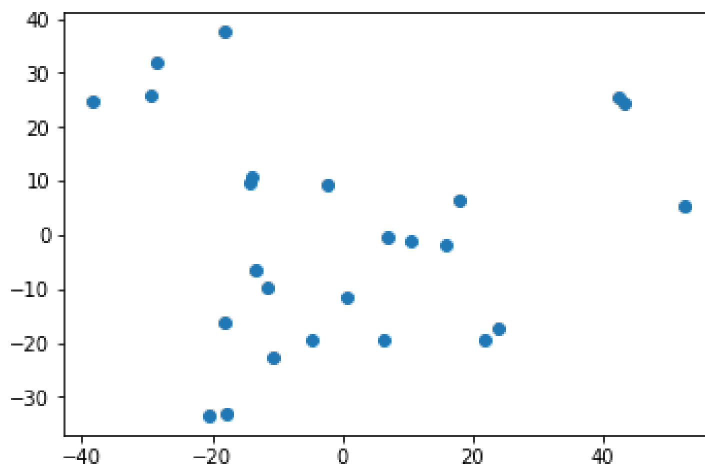
## 3.6 Clustering

One hypothesis that emerged during this thesis was that similar cells could be trained together to improve model performance and to counteract the lack of data given the discontinuous sampling constraint. Thus, clustering was explored with two main methods: Principal Component Analysis (PCA) and Uniform Manifold Approximation and Projection (UMAP). PCA (Pearson, 1901 and Hotelling, 1933)[Pea01, Hot33] and UMAP (McInnes, et. al., 2018) [MHM18] are both dimensionality reduction techniques and part of the unsupervised learning algorithms. The idea is that data can be represented in a more compact form (with less dimensions) while keeping some of its properties (global distance between features, variance, etc.). PCA transforms data with a high number of dimensions into a lower dimensional representation. These lower dimensions are linearly uncorrelated and they are known as *principal components*. The first principal component summarizes most of the data's variance, and each of the following ones summarize as much as possible of the remaining variance. However it does not capture non-linear structure.

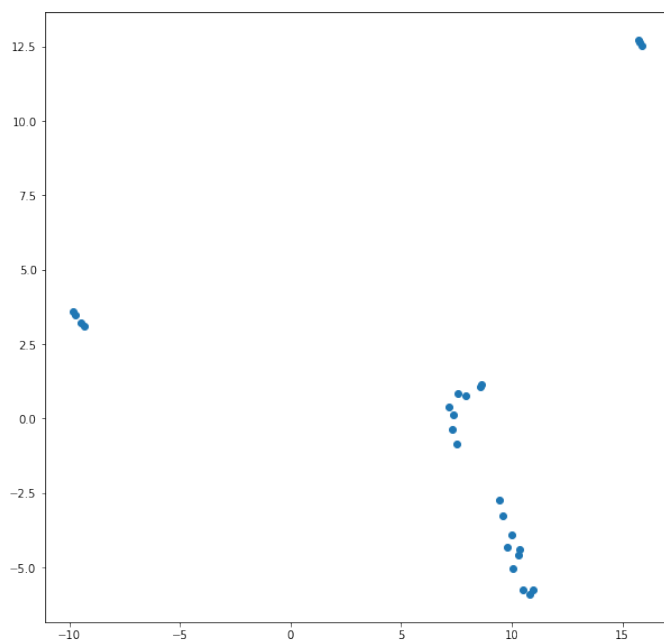
Uniform Manifold Approximation and Projection (UMAP) was derived from manifold learning methods and topological data analysis. An important advantage is that it can reduce dimensionality while preserving global (linear and non-linear) structure better than similar approaches, such as t-SNE and PCA.

During initial clustering attempts, PCA was the first option to test given its simplicity and widespread use in machine learning tasks. Nonetheless, after plotting the two main components, there was no structure (defined clusters) to be found (Figure 13a).

I was then advised by Nokia experts to apply UMAP, which as can be seen in Figure 13b had more defined clusters. Other methods that were explored were K-Means, X-Means, and Factor Analysis.




(a) PCA




(b) UMAP

Figure 13: Clustering by Dimensionality Reduction. The above plots show the two main components resulting from applying PCA and UMAP to the data from the 25 viable radio network cells. Each blob represents a cell, and the distance between each of them represents their similarity. Thus, clustering can then be performed based on this measurement (e.g. with k-nearest neighbours or DBSCAN).

## 3.7 Embeddings

Another emergent hypothesis was that timestamp values could be valuable for the model to learn time-dependent patterns. Embeddings are a common and effective way of translating these kind of values into float numbers that can be computed by deep learning models. Embeddings represent concepts (e.g. words, symbols, dates) as vectors of real numerical values. As opposite to one-hot encoding, embeddings keep the relationship between variables (concepts) and represent it as distance. For embeddings, distance is the result from vector difference. Thus, distance will be closer for concepts which are similar and further for those which are different or unrelated.

For the experiments presented in this thesis, embeddings were obtained via the Keras embedding layer. An embedding's output size is arbitrary, however, there are existing heuristics that can be considered. For this thesis, I applied the heuristic from [How19] based on [GB16], which recommends to divide the cardinality of each categorical variable by two and then add one to acquire the embedding size per categorical variable. The embedding parameters can be observed in Table 3. As it will be demonstrated in Section 5.3 in the end, they did not improve the model performance.

Table 3: Embeddings Parameters. All dimensions (dim) refer to the maximum number of possible values that the variable con have. The total size of the resulting concatenated embedding vector was 30

| **Parameter** | **Input dim** | **Output dim** |
| --- | --- | --- |
| ID | 25 (max number of sources) | 13 |
| hour of day | 24 (max hours in a day) | 13 |
| day of week | 7 ( max days in a week) | 4 |

## 3.8 Data Generators

Data generators are useful constructs (commonly defined as a Class) for automating batch generation for training, validation and test sets. When training or evaluating a model, the batch generator will iteratively transfer it the required (training, validation or test) batches. When using a data generator, we can specify whether the data samples will be transferred to the algorithm in order or shuffled, the latter

usually applied for training generalization. Other aspects that can be specified are training, validation and test splits, as well as batch size. Together with a gradient-based optimizer's learning rate, batch size can impact the variation in the sample distribution, memory usage, and training speed. Larger batches will represent better the data distribution but will take longer to compute and will require a larger amount of memory [GBC16, p. 279]. According to Wilson and Martinez (2003) [WM03], mini-batches will be helpful in adding regularization, yet will require more time as well, since more batches will be needed to train for one epoch, that is, to cover the full training set. Therefore, one should evaluate with different batch sizes before determining the most suitable for the task.
With the previous in mind, and in order to have greater variety and control over the data samples, data generators were applied for my experiments (code provided by Nokia's MN SoC R&D ML&AI team). These generators provided data splitting and batch generation. After splitting the datasets, shuffling was applied only for the training set. An important characteristic of these particular generators was producing target sequences of the target KPI, together with input sequences of their corresponding input features (KPIs used to predict the target). These sequences would then be submitted as training samples to the model (described in more detail in Section 3.9.3).

## 3.9 Learning Scenarios

Going further from splitting the data into training and evaluation sets, different learning scenarios were explored by splitting it with respect to its cells. In this section the following scenarios will be presented: centralized learning, cluster-wise learning, cell-wise learning. Except for centralized learning, the other two cases can be applied in federated learning, so at the end of the section the procedure for this will be shown.

### 3.9.1 Multi-cell Scenarios

Centralized learning was chosen as the opposition for de-centralized learning. It consisted of only one neural network model which was trained and tested with data from all cells. In order to have a uniform comparison, for this and the following scenarios, each cell's data was tested independently.
Cluster-wise learning was an alternative between centralized and cell-wise learning,

with the addition of trying the hypothesis that cells with similar time series should better be trained together. Clusters were selected as described in Section 3.6. Then, one model was trained per each cell cluster with its corresponding cells' data. Testing was performed in the same manner.
Cell-wise learning was also explored given that it is the extreme case of de-centralized learning and that on the federated learning setting, cells were going to be trained individually as well. Each cell was trained with its corresponding data and evaluated in the same fashion.

### 3.9.2 Federated Learning Scenario

With the intuition obtained from the previous scenarios, a series of federated learning cases were attempted. As a starting point, the federated update period was to be found by comparing short, mid and long-term updates.
Once having established this parameter, two cases were to be compared. The original or naive case was the first one. There was a model per each cell in addition to the global shared model. In the first phase, data was fed to each cell depending on the update size (daily or weekly updates being equivalent to 96 and 672 timesteps respectively), so each would train and produce a new update to their model. In the second phase, all model updates (weights) were sent to the federated averaging algorithm for updating the global model. The last phase consisted of sending the global model back to each cell and repeating the process with the consequent daily or weekly slice of data.
The second case was to evaluate federated learning in a cluster-wise manner, thus having a shared global model per each cluster. For both cases, testing was performed individually for each cell.

### 3.9.3 FL Transfer Scenario

Since cluster-wise learning proved to be the most beneficial for the telecom scenario (See Sections 5.4 and 5.5.3), transfer learning of a cluster's federated model to new cells was assessed as a final experiment. The following steps were performed on the most populated cluster from Section 5.5.3:

- 25% of its cells were picked at random from within the cluster (as test sets).

- One federated model was trained per each of the previously selected cells with each one of them kept aside at a time, thus, producing different federated models.
- For each test cell, the federated learning model trained in its absence was transferred to it.
- Then, each cell was evaluated (without further training) to get the starting performance of which a new cell would benefit if a (cluster) federated model was transferred to it.
- Next, these models were trained with a day of their corresponding cell's data at a time to simulate the development of their performance at deployment. The validation set was 20% of records from the following day.
- In a separate instance, an untrained model was generated for each test cell and as in the previous step these models were trained and evaluated on a daily basis.

Finally both cases, with and without transfer learning were compared.

For each learning setting described in this section, the global model's neural network weights were initialized with the *Xavier Initialization* [GB10] method and then these were replicated for all cells. The input data received from the generators was a concatenated time series, defined by input length (see Figure 14). This means how much into the past we look to predict the future (e.g. 2 hours = 8 timesteps, 1 day = 96 timesteps). It is important to notice that the update step and input length are not the same. Update step refers to the total amount of data to be fed to the neural network before updating the global model, and input length is the amount of concatenated data that is fed as one training sample. Moreover, before a global model update was achieved, several training samples were passed to the neural network to train with.

Figure 14: Input Data

As step predictions were performed, the output of the neural networks was also a concatenated time series containing the target KPI's values. The forecast was defined by the desired output length (See Figure 15). This means how much into the future we want to predict.

Figure 15: Ouput Data

# 4 Models

Models are a way of explaining different phenomena. In mathematics, a model is the relationship between one or more independent variables and one or more dependent variables. For time series, it is the relationship between the past values of different kinds of variables (time dependent and independent) and the future values of one or more time-dependent variables. Some examples of machine learning models that satisfy the necessary conditions for federated learning (Section 2.3) are: linear and logistic regression, neural networks and support vector machines [Har18]. Neural networks were selected for the federated learning scenario, as they are parametric, they can solve complex non-linear problems and weights are easy to extract, which is a relevant quality for performing the *federated averaging* algorithm.
In this section the KPI modelling approaches will be described. Section 4.1 will illustrate how the base performance for time series prediction was computed via a simple (non-machine learning) time series prediction model called *persistence model.* Neural networks and two of their architectures were already presented in Section 2.2.3, so the evaluation procedures or metrics, as well as the optimization algorithms will be depicted in Sections 4.2 and 4.3. Last but not least, the neural network hyper-parameters and their tuning procedures will be presented in Sections 4.4 and 4.5.

## 4.1 Baseline Model

A baseline is the starting point when comparing and/or optimizing solutions. The most common baseline model for time series is the persistence model. As the name suggests, it "persists" (repeats) the last observed value from 1 (See Equation 11) to $n$ times for an *n-step* forecast (See Equation 12). The persistence model is a simple non-machine learning model, so in that sense, when I compared its performance against the upcoming machine learning models, it helped to determine whether machine learning is a feasible and relevant approach for solving the task of telecom KPI prediction.

$$x_{t+1} = x_t \tag{11}$$

$$x_{t+1:t+n} = [x_t \text{ for } i = 0; i < n; i++] \tag{12}$$

## 4.2 Metrics

Metrics are the way in which the algorithms' performance is measured. For the experiments related to this thesis, the following were selected:

- **Mean-Squared Error (MSE):**

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{13}$$

- **Root Mean-Squared Error (RMSE):**

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{14}$$

- **Mean Aboslute Error (MAE):**

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{15}$$

- **Mean Absolute Percentage Error (MAPE):**

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right| * 100 \tag{16}$$

MSE (Equation 13) was selected for training the models and MAPE (Equation 16) was selected for evaluating and comparing them. The reason for choosing MSE for training was that it is a smooth and convex function, which makes it easier to minimize the gradient. In contrast to MAE and MAPE, MSE penalizes large errors more than smaller ones given the square of the error which is desirable for optimal predictions. Some disadvantages are, that it is sensitive to outliers, which in this dataset could be considered as the spikes (discussed further in Section 5.4) and that it underestimates the error when targets and predictions have small magnitudes. MAPE has the advantage that it is a relative error with direct interpretation, so we could understand how distant was the predicted value from the target value with a standardized unit (percentage). This was needed to compare the experiments' results given the different data aggregations and update settings in which the models were trained. It should be noted that MAPE has instability when the true (or target) value is small, but this was rarely the case for the target KPI.

## 4.3 Optimizers

Optimizers in machine learning are algorithms that shift the parameters towards fitting the target or objective function. Based on the gradient descent criteria, the following optimizers were tested:

- **Stochastic Gradient Descent (SGD)** In contrast to gradient descent, which runs over all the training data for the update, SGD runs over a random small portion of the dataset or mini-batch at a time (See Algorithm 2). This accelerates the learning process and provides regularization [WM03].
- **RMSProp** RMSProp is an adaptive learning optimizer. This means that each parameter has its own learning rate, which is adapted while training. RMSProp does so by scaling the learning rates inversely proportional to a factor of the historical squared gradient values $-\frac{\epsilon}{\sqrt{\delta+r}}$ (See Algorithm 3). Thus, the update will weigh more on the direction of smaller gradients, providing smoother updates. A similar optimizer, AdaGrad, does the same, but it does not perform well in non-convex settings. RMSProp could be thought of as an improved version of it [GBC16, pp. 307-308].
- **Adam** Adam is another popular adaptive learning optimizer. It includes first and second order momentum terms, together with their bias correction (See Algorithm 4). Then, it divides the first moment by a factor of the second one to scale the learning rates, similarly to RMSProp [GBC16, pp. 308-309].

**Algorithm 2:** Stochastic gradient descent (SGD) update at training iteration $k$. Source [GBC16].

**Require:** Learning rate $\epsilon_k$
**Require:** Initial parameter $\boldsymbol{\theta}$
**while** *stopping criterion not met* **do**
  Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}$
**end**

**Algorithm 3:** The RMSProp algorithm. Source [GBC16].

**Require:** Global learning rate $\epsilon_k$, decay rate $\rho$.

**Require:** Initial parameter $\boldsymbol{\theta}$

**Require:** Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small numbers.

Initialize accumulation variables $\boldsymbol{r} = 0$

**while** *stopping criterion not met* **do**

Sample a minibatch of $m$ examples from the training set $\left\{x^{(1)}, ..., x^{(m)}\right\}$ with corresponding targets $\boldsymbol{y^{(i)}}$.

Compute gradient : $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y^{(i)}})$

Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$

Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. `//` $-\frac{1}{\sqrt{\delta+\boldsymbol{r}}}$ `applied element-wise`

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

**end**

**Algorithm 4:** The Adam algorithm. Source [GBC16].

**Require:** Step size $\epsilon_k$ (Suggested default: 0.001).

**Require:** Exponential decay rates for moment estimates, $\rho_1$ and $\rho_2$ in [0,1). (Suggested defaults: 0.9 and 0.999 respectively)

**Require:** Small constant $\delta$ used for numerical stabilization. (Suggested default: $10^{-8}$)

**Require:** Initial parameters $\boldsymbol{\theta}$

Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$

Initialize time step $t = 0$

**while** *stopping criterion not met* **do**

Sample a minibatch of $m$ examples from the training set $\left\{x^{(1)}, ..., x^{(m)}\right\}$ with corresponding targets $\boldsymbol{y^{(i)}}$.

Compute gradient : $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y^{(i)}})$

$t \leftarrow t + 1$

Update biased first moment estimate $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1) \boldsymbol{g}$

Update biased second moment estimate $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2) \boldsymbol{g} \odot \boldsymbol{g}$

Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$

Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

Compute update: $\boldsymbol{\Delta\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}} + \delta}}$ (operations applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{\Delta\theta}$

**end**

## 4.4 Hyper-parameters

Hyper-parameters, in contrast to parameters, are not learned from the data, rather they are tuned by whoever performs the experiment. They also affect on how the learning algorithm will learn, for example, learning speed, model complexity, and over-fitting prevention. The following are the neural network hyper-parameters that were selected for tuning:

- **Input sequence length** It refers to how many steps into the past are sent into the algorithm.
- **Max number of epochs** During naive experimentation overfitting occurred, mainly with LSTM. Therefore, Keras's EarlyStopping callback was added with a fixed patience of ten. This means that the algorithm would stop running epochs if after ten of them the validation error had not improved. Then, *max number of epochs* would be the upper bound for the algorithm run (in case Early Stopping was not called).
- **First layer's number of units** For simplicity, only the first NN layer's number of units could be defined explicitly. If more than one layer was present, each of the following ones would have the previous number of units halved.
- **Number of layers** The number of NN layers before the output layer.
- **Optimizer** Optimization algorithm which shifts the parameters towards their optimal values (e.g. Adam, RMSProp, SGD).
- **Activation function** It is a non-linear function $g$ applied element-wise within a neural unit, usually after a an affine function (See Equation 17). Cybenko (1989) proved that any continuous function can be approximated with a two-layer (1 hidden layer) neural network, if the activation function is non-linear [Cyb89].

$$h_i = g(x^T)W_{:,i} + c_i \tag{17}$$

## 4.5 Hyper-parameter Tuning

Three methods were considered to optimize the performance of the neural networks:

- **Grid Search** This method explores all possible combinations from a set of values (in this case, hyper-parameter values) and after all of them are evaluated, the best one can be selected. It can take up to $n^k$ evaluations where $n$ is the size of the largest set of test values for an hyper-parameter and $k$ is the number of hyper-parameters.
- **Guided Search** (a.k.a. *coordinate descent*) Starting from an arbitrary configuration, one hyper-parameter is tuned at a time. This is done for all hyper-parameters and each time, the best configuration achieved so far is saved.
- **Bayesian Optimization** By means of the Bayes Rule this method shifts hyper-parameters values in the direction of highest probability of improvement. It begins by fitting a Bayesian statistical model to the data based on the target function. Following this, an acquisition function, calculates the expected improvement of candidate points, given the posterior distribution [FH19, Fra18]. Hyper-parameter candidate values can now be continuous and their range is provided instead.

# 5 Results and Discussion

In this section, the results from various experiments will be presented. These will range from how was the model selected until the evaluation of federated learning and the transferring of a federated model to new radio cells. Section 5.1 will begin by presenting the baseline model performance. Subsequently, the machine learning model selection and tuning outcomes will appear in Section 5.2. Afterwards, the embeddings alternative will be depicted in Section 5.3. Section 5.4 will compare different learning scenarios for this KPI forecasting task from telecom cells. Finally, in Sections 5.5 and 5.6 the outcome of federated and transfer learning experiments will be presented and discussed.

## 5.1 Baseline Performance

As a starting point, the *persistence model* was applied to the dataset in order to forecast the target KPI. Recalling from Section 4.1, this model is one of the least complex ways to tackle time series and it does not apply statistical nor machine learning methods. For this and for further experiments, the forecast lengths tested were 15mins, 1 hour and 6 hours with MAPE as metric.

Table 4: Baseline Model MAPE

| **Forecast Length** | **MAPE** |
|---|---|
| 15 mins | 36.59 |
| 1 hour | 49.49 |
| 6 hours | 126.21 |

By observing the results from Table 4, it can be observed that long-term predictions are unreliable, since the mean average error surpasses the predictions magnitude (the predictions are off by 126% of the target's value).

## 5.2 Model Selection and Hyper-parameter Tuning

After knowing the base performance to achieve so that the following machine learning models were significant, neural networks were evaluated. Neural networks were selected as the model architecture for federated and other machine learning settings,

because of the requirements described in section 2.3, to account for non-linearity, and because it was the method applied in the original paper [KMY+16], thus to have a proven and valid reference. Within neural networks, two of their most frequently used configurations for time series prediction were chosen: Multi-Layer Perceptron (MLP) and Long Short-Term Memory (LSTM).

In order to have an objective comparison, each configuration's hyperparameters were tuned. Then, for evaluation, the performance was measured with MAPE for short-term, mid-term and long-term target KPI forecast. These time periods were defined by Nokia according to the business case and their KPIs domain knowledge; more specifically: predicting the next fifteen minutes, the next hour and the next six hours.

As mentioned in Section 4.5, three methods were contemplated to tune the hyperparameters defined in Section 4.4 for MLP and LSTM. Because of the computational complexity to perform a Grid Search (9216 hyper-parameter combinations per model architecture type), and because Bayesian Optimization didn't converge during the maximum run-time at the server (7 days), Guided Search was the approach to perform hyper-parameter tuning. However, this was not the best choice as it will be discussed in Section 7.

The test values were the following:

Table 5: Hyperparameter test values

| **Hyperparameter** | **Test Values** |
|---|---|
| Model type | MLP, LSTM |
| Sequence length | 8, 24, 48, 96 |
| Max epochs | 25, 50, 100 |
| Batch Size | 32, 64, 128, 256 |
| First layer's units | 32, 64, 48, 96 |
| Number of Layers | 1, 2, 3, 4 |
| Optimizer | rmsprop, adam, sgd |
| Activation Function | tanh, relu, leaky relu |

Early stopping with default patience 10 was applied since LSTMs were overfitting.

Table 6 presents the results from the hyper-parameter selection. From all hyper-parameter combinations used during the search, MLP obtained lower average MAPE

and runtime than LSTM, so it became the chosen configuration. From the LSTM results, it can be observed that it got slightly better short-term performance and considerably worse long-term performance in comparison with MLP. Makridakis, et. al. (2018) [MSA18] confirmed this behaviour. On their paper they reported that indeed, LSTMs fit the data better than simpler models like MLP, but they will not have as good future predictions. Also, the results from both configurations in Table 6 show significantly lower MAPE than the baseline model for mid and long-term predictions, and for short-term a relatively small difference of $\pm 2\%$ MAPE. Therefore, it was safe to proceed with neural networks for further experiments.

Table 6: Hyperparameter Selection. This table describes the MLP and LSTM configurations with lower average MAPE, including the hyper-parameter combination, the MAPE per each forecast period, number of epochs until convergence, runtime and average MAPE (over these three forecast periods)

| model type | **MLP** | **LSTM** |
|---|---|---|
| sequence length | 96 | 24 |
| output length | 24 | 24 |
| max epochs | 50 | 50 |
| units Layer1 | 64 | 256 |
| number of layers | 2 | 2 |
| optimizer | adam | adam |
| activation | leakyrelu | relu |
| batch size | 64 | 128 |
| MAPE 15 mins | 38.76 | 37.16 |
| MAPE 1 hour | 41.44 | 41.25 |
| MAPE 6 hours | 46.83 | 55.03 |
| total epochs | 38 | 21 |
| Runtime (seconds) | **275.08** | **1295.18** |
| Average MAPE | **42.35** | **44.48** |

## 5.3 Embeddings

In order to capture the time dependent features (in this case, the day of the week, hour of the day and cell ID), embeddings were proposed as a solution. To assess whether embeddings improved the model's predictions, MLP was tested with and

Table 7: MAPE results with and without the embedding layer. In this table the MAPE resulting from adding the embedding layer to the individual or cell-wise models as well to the centralized model is reported. This was done for the three previously defined forecast lengths

| Learning | Embeddings | Forecast Length | MAPE mean | std | min | max |
|---|---|---|---|---|---|---|
| Cellwise | Yes | 15 min | 28.45 | 5.75 | 17.59 | 42.67 |
| | | 1 hour | 29.67 | 5.86 | 19.06 | 40.65 |
| | | 6 hour | 31.60 | 6.38 | 20.95 | 44.65 |
| | No | 15 min | 28.94 | 6.25 | 18.12 | 41.41 |
| | | 1 hour | 30.11 | 6.59 | 18.82 | 44.28 |
| | | 6 hour | 31.61 | 6.61 | 20.86 | 44.73 |
| Centralized | Yes | 15 min | 38.26 | 7.81 | 20.57 | 55.82 |
| | | 1 hour | 41.34 | 8.55 | 22.21 | 57.44 |
| | | 6 hour | 45.48 | 9.83 | 25.01 | 61.17 |
| | No | 15 min | 38.55 | 8.05 | 19.73 | 54.63 |
| | | 1 hour | 41.41 | 8.33 | 22.11 | 55.55 |
| | | 6 hour | 47.43 | 9.04 | 29.33 | 63.23 |

without the embedding layer for centralized and per cell learning.

From the results in Table 7 it can be observed that there was no significant difference in MAPE by adding the embedding layer. This observation applied for both learning configurations, so in order to optimize neural network's architecture, the embedding layer was not kept for future experiments.

## 5.4 Multi-Cell Learning

Having settled the model's architecture, different learning scenarios were evaluated as described in Section 3.9.1. The first one was centralized learning, in which there was only one model and all the data from all cells was used to train it. The next one was, per-cell (or cell-wise) learning, in which there was a model per each cell and each was learned only from that cell's data. Finally, with clusters obtained by applying UMAP, cluster learning was performed, meaning that a model was obtained per each cluster.

For all scenarios, the network hyperparameters remained the same, as in Table 6. Each cell's evaluation was compared within learning scenarios as it can be observed in Figure 16.

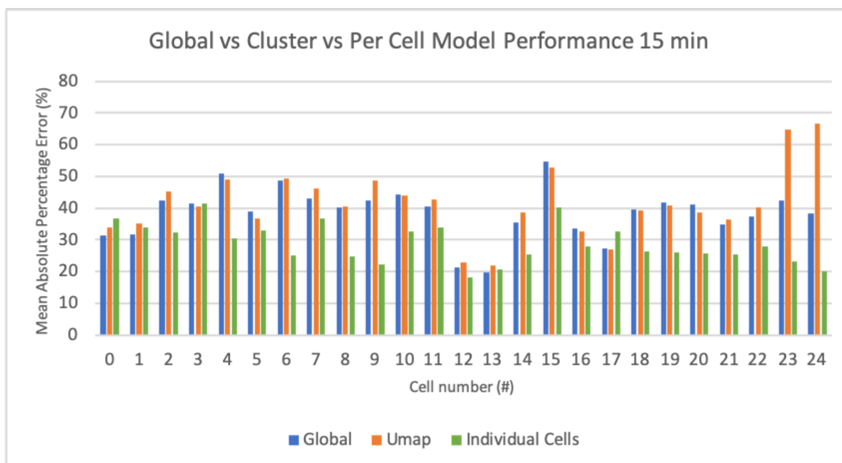
From these results, it could be noticed that training cells individually will provide lower MAPE, contrary to a centralized model that combines all observations at once. However, as a starting point for a new cell, the centralized model provides a head-start of less than 50% MAPE (for the first two output length options). Moreover, clustering cells to generate several "global models" can provide an even better starting performance. Therefore, I concluded that this configuration would be more beneficial for the telecom transfer-learning objective. My hypothesis is that providing a federated model trained from a cluster of similar cells to the new ones will reduce the time to learn their model as they gather and learn from their own data. Afterwards, training by themselves will provide the best forecasts, as was proven for the per-cell configuration.

Furthermore, the KPI time-plots were analysed to obtain the relationship between the learning setting and its performance against the others. My observations were the following:

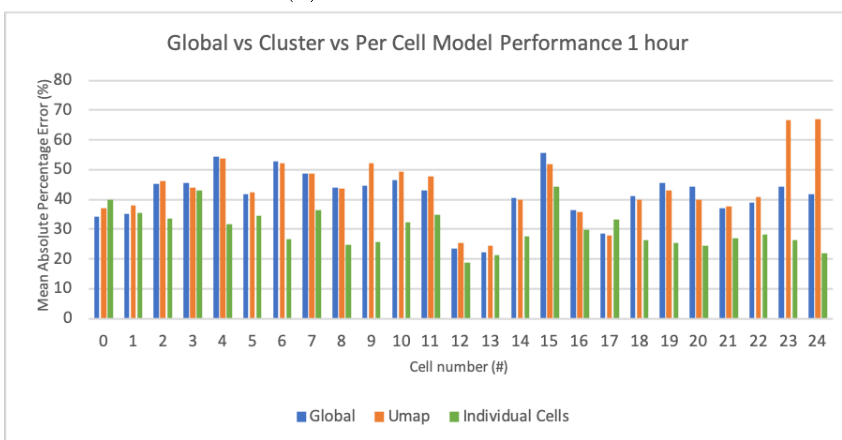
- In cells with spikes considerably larger than the pattern, MAPE was high for all settings. See Figure 17a and in Figure 16 notice cells 4, 6 and 15.
- MAPE was similar for all settings when the autocorrelation of the predicted KPI was high (closer to 1 with respect to other cells). See Figure 17b and in Figure 16 notice cells 1, 12 and 13.
- Cell-wise learning outperformed the other settings when the pattern shifted in shape or magnitude. In addition, the auto-correlation decreased faster. Thus, more local data was needed to adjust the model rather than from other cells. See Figure 17c and in Figure 16 notice cells 9, 23 and 24.

I suggest that being able to identify these behaviours beforehand (spikes and high, low, constant, increasing, or decreasing autocorrelation) could enable better results not only for federated learning, but in general for radio networks time series prediction.

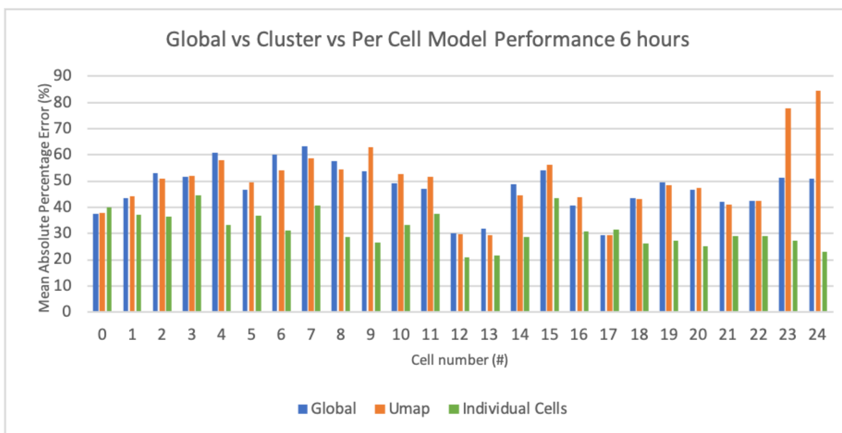
Other clustering methods were also evaluated resulting in very similar MAPE ($\pm 1.4\%$ average MAPE difference) as achieved with UMAP (See Appendix A).




(a) 15 minutes forecast

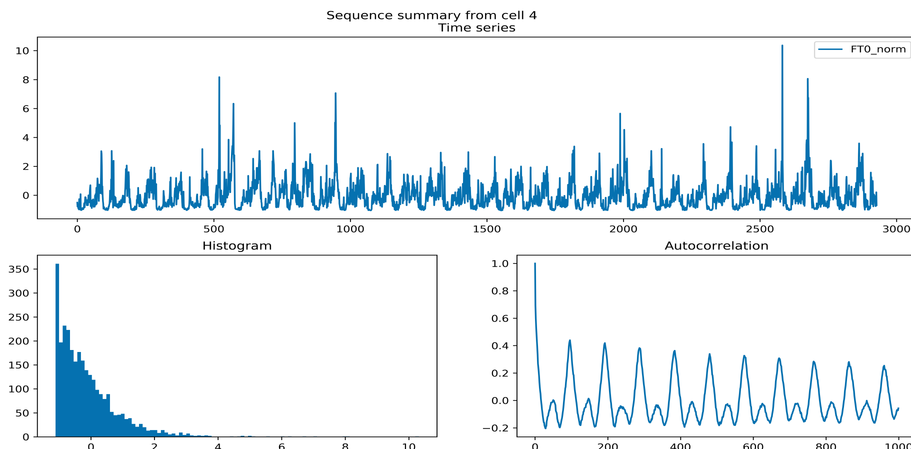



(b) 1 hour forecast

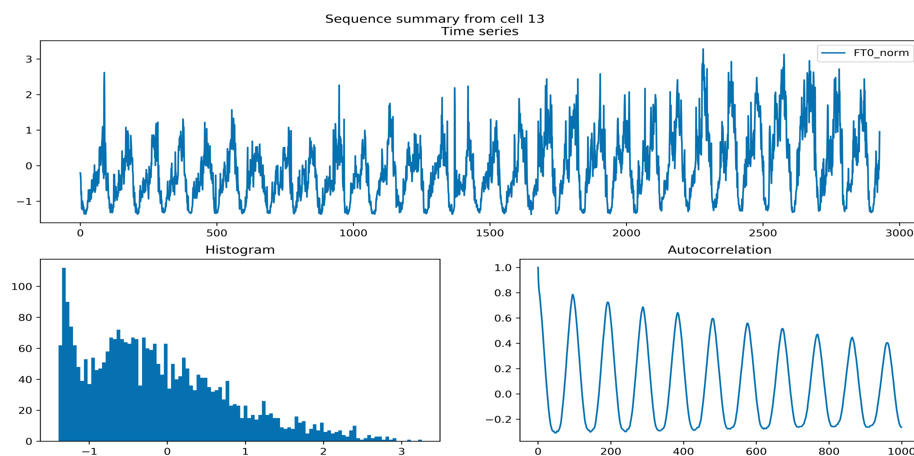



(c) 6 hours forecast

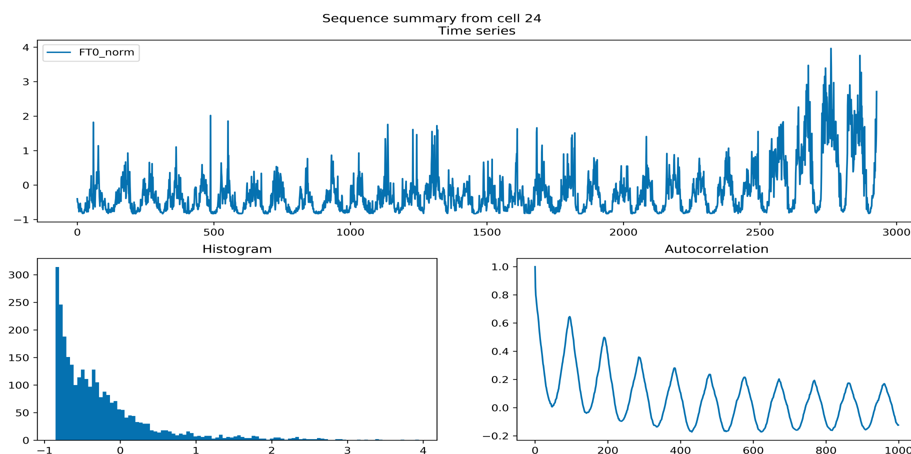
Figure 16: Multicell forecast of target KPI




(a) Example of high MAPE for all learning settings




(b) Example of similar MAPE for all settings




(c) Example of cell-wise MAPE better than centralized and cluster-wise

Figure 17: Multicell Forecast Analysis. Visualization produced with code provided by Nokia's MN SoC R&D ML&AI team

## 5.5 Federated learning experiments

### 5.5.1 Federated Updates

For evaluating the federated learning setting, the update period had to be established. From Nokia's domain experts, I was recommended that radio networks' KPI behaviour changes drastically between months. Then, three different update periods' outcomes were compared: daily, weekly and full-training-dataset updates (See Figure 18). The reason for the last one was to test the case of updating the model with the largest of the available data segments (in this case 5 weeks).

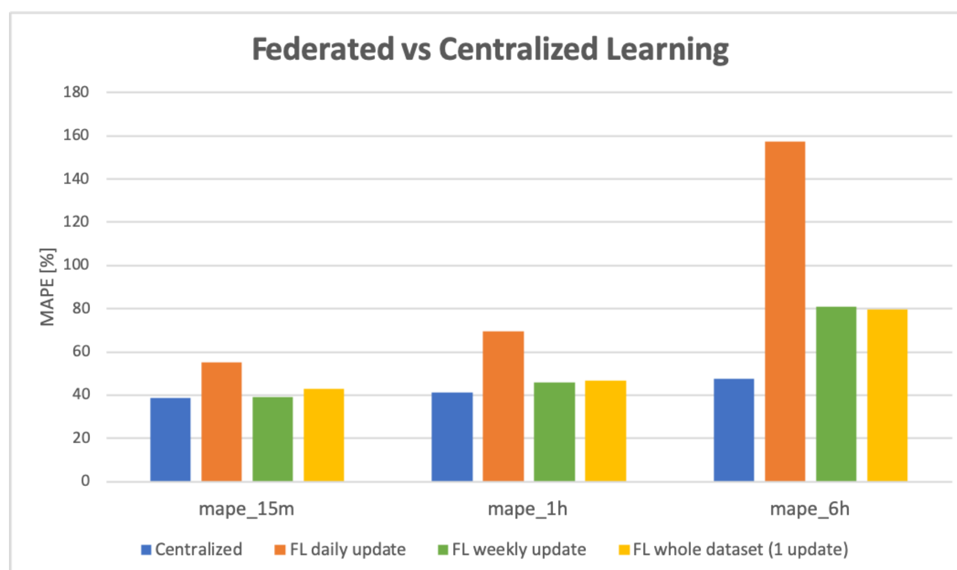



Figure 18: Federated Update Periods and Centralized Learning MAPE

### 5.5.2 Data Transfer

At the same time, update sizes and data size to be transferred were compared respectively for federated and centralized learning (See Figure 19). By observing these results combined with the similar performance between weekly-update federated and centralized leaning, I can confirm that federated learning can have MAPE equivalent to the one from the centralized model for short and mid-term predictions, while minimizing data transfer.

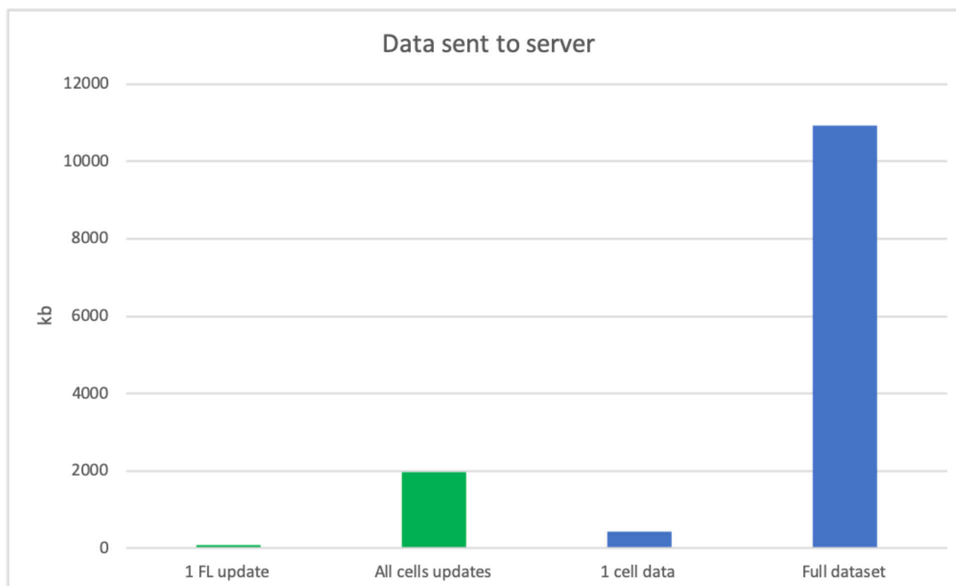
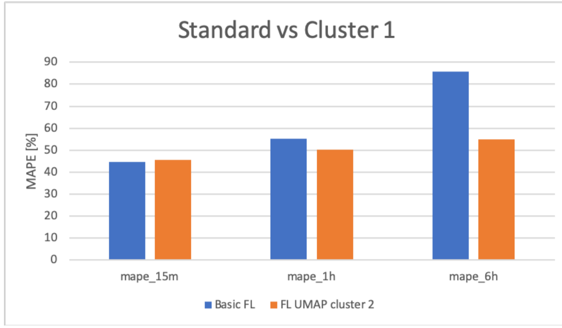


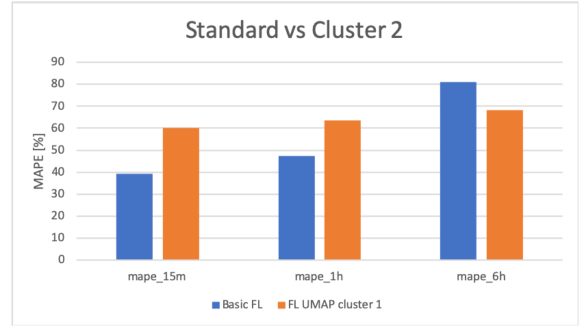

Figure 19: Data transfer sizes for Federated and Centralized Learning

### 5.5.3 Naive FL vs cluster-wise FL

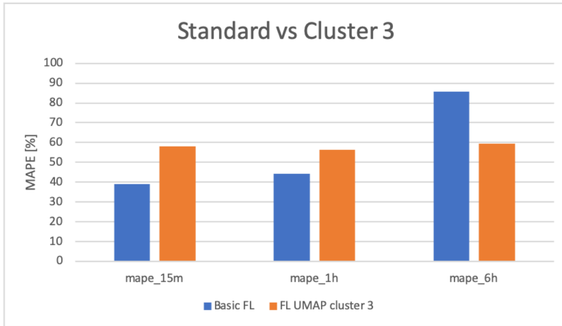
Afterwards, weekly updates for one federated model for all cells, as well as for one federated model per cluster scenarios were analysed. Again, testing was done per each cell and compared accordingly. From the results in Figure 20 it can be noticed that clustering similar cells has the potential for further improving the default federated learning setting's performance, like in Figure 20 (a). Also, for the long-term forecast, cluster-wise federated learning overcame its default version for all clusters.



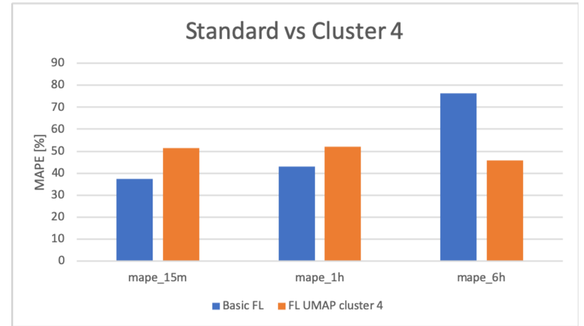

(a) FL: standard vs cluster1




(b) FL: standard vs cluster2




(c) FL: standard vs cluster3




(d) FL: standard vs cluster4

Figure 20: FL: Standard vs Clusterwise learning

## 5.6 Transfer Learning Score

As a last experiment, the performance from transferring a federated model to new cells was explored, following the steps described in Section 3.9.3. For the purpose of statistical significance, the largest cluster (cluster 4, consisting of 11 cells) was selected. The random test cells were 3, 16, and 22 (approximately 25% of the cluster's cells).

The starting performance from transferring the federated model obtained from the rest of the cells can be observed from Table 8. When compared to the average baseline performance of the centralized model (See Section 5.1) we can notice that for short-term predictions the baseline MAPE was lower by 5% and for middle-term it was roughly the same. For long-term predictions, however, the transferred federated model's starting performance MAPE was 59% lower, which also makes it viable in comparison to the baseline (which surpassed 100% MAPE). This evidence shows that federated learning is already beneficial for telecom's problem of modelling

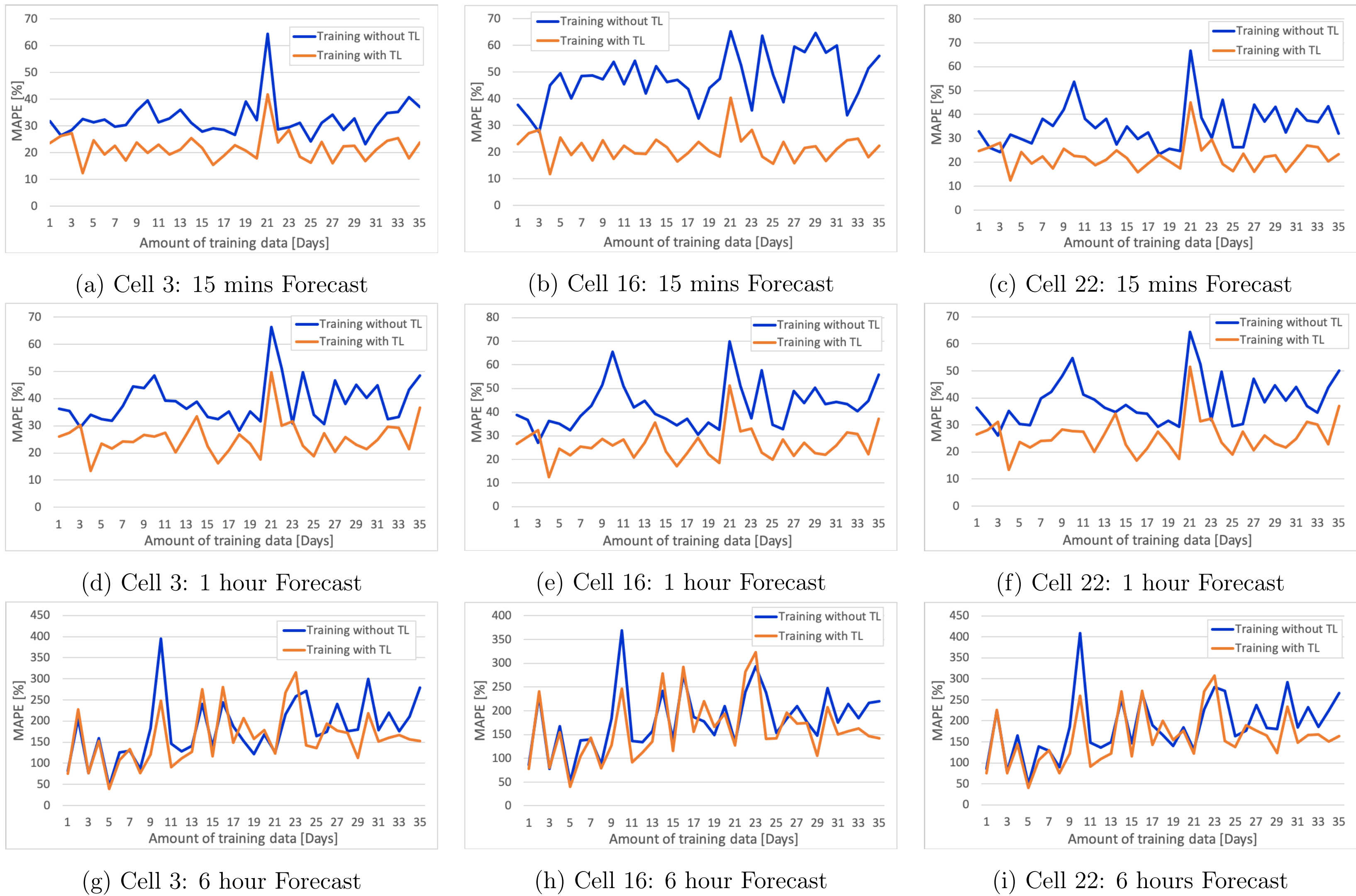
new cells.

Table 8: Transferred FL model starting MAPE

| cell ID | mape 15min | mape 1h | mape 6h |
|---:|---:|---:|---:|
| 3 | 42.88 | 50.55 | 67.09 |
| 16 | 38.58 | 48.19 | 66.79 |
| 22 | 43.12 | 50.58 | 67.12 |
| Average | 41.53 | 49.77 | 67.00 |

Next, these trained federated models were transferred to new untrained models corresponding to the new cells which would be added to the network. Afterwards, the transferred models were trained with data from their corresponding cell's time series. Initially, I would have trained them with weekly updates as it was proven to be better in Section 5.5. However, to compare with more granularity the MAPE evolution of transfer learning against its opposite scenario (without transferring the model), the test cells were trained and evaluated on a daily (dataset) basis. This means that the amount of data that was used per update was equivalent to one day of data samples.

With regards to the opposite scenario or transfer learning baseline model, training was performed from scratch for each of the three test cells on a daily basis and tested in the same way as for the transferred model (with 20% of the following day's data). In Figure 21 the comparison from transferring a federated model or not can be regarded. This resulted in the transfer learning outperforming training the new cells from scratch, for short and mid-term predictions; for the long-term, the performance was relatively similar.




(a) Cell 3: 15 mins Forecast

(b) Cell 16: 15 mins Forecast

(c) Cell 22: 15 mins Forecast

(d) Cell 3: 1 hour Forecast

(e) Cell 16: 1 hour Forecast

(f) Cell 22: 1 hour Forecast

(g) Cell 3: 6 hour Forecast

(h) Cell 16: 6 hour Forecast

(i) Cell 22: 6 hours Forecast

Figure 21: Transfer Learning Impact. In these plots the Test MAPE between cells trained with and without Transfer Learning is compared. Training was performed with daily data batches and testing was performed after each update.

In Table 9 the final MAPE scores are shown. The transferred model was able to outperform the TL baseline model's final scores as well as the centralized learning baseline model's; in this last case, with the exception of the long-term prediction, in which for both, it was not viable for telecom deployment (greater than 100% MAPE).

Table 9: Last day evaluation MAPE. This table displays the MAPE scores for each of the three test cells in both settings: with and without TL. The scores shown are the ones obtained when testing after the last day of training, as well as the average.

| | Without TL (baseline) | | | With TL | | |
|---|---|---|---|---|---|---|
| cell ID | mape 15min | mape 1h | mape 6h | mape 15min | mape 1h | mape 6h |
| 3 | 36.95 | 48.38 | 278.92 | 23.66 | 36.70 | 152.60 |
| 16 | 56.04 | 55.83 | 220.40 | 22.44 | 37.03 | 142.35 |
| 22 | 32.08 | 50.10 | 265.33 | 23.35 | 37.02 | 164.11 |
| Average | 41.69 | 51.44 | 254.88 | 23.15 | 36.92 | 153.02 |

# 6 Conclusion

From the initial experiments in Sections 4.5 and 5.2 it was confirmed that neural networks, particularly MLP, can contribute to predict the future of radio networks' KPIs. Furthermore, in Section 5.4 the advantages of cell-wise, global and cluster-wise learning were revealed. Then, the federated learning experiments in Sections 5.5 and 5.6 provided evidence that for telecom radio networks' KPI prediction, this setting can have almost equivalent performance as the centralized one, with the added advantage of significantly reducing data transfer. Moreover, clustering similar cells together to train as federations and using their global model to train new cells proved to be better than training them from scratch. It should be noted that clustering will not always be better than a single federation as it was observed in Section 5.5.3, but if correctly tuned, clusters can obtain better performance.

Last but not least, in Section 5.6 the initial hypothesis about reducing the training time to learn a new radio cell's model by means of transfer learning of a pre-trained federated model was confirmed.

Given the results and analysis from this thesis, I conclude that the de-centralized learning of radio networks data via federated learning is not only feasible, but it also has significant advantages as the following:

- If having a federated model available, transferring it to a new cell will reduce several days of training for this cell, contrary to doing it from scratch.
- In contrast to centralized learning, the federated setting will considerably reduce the amount data transfer, and it is possible to keep an equivalent performance as with its centralized counterpart.

# 7 Future Work

One main point of improvement would be to compare the outcome of the guided search against the last result from the Bayesian Optimization for the hyper-parameter tuning, as iteration convergence is not a requisite for achieving at least a sub-optimal result. Sources such as *Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures*[BYC13], *Practical bayesian optimization of machine learning algorithms* [SLA12], *Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms* [THHLB13], and *Meta-surrogate benchmarking for hyperparameter optimization* [KDH+19] confirm how superior this algorithm is in contrast to optimization performed by experts (where guided search can be found) as well as previous commonly used algorithms (e.g. Differential Evolution, Covariance Matrix Adaption Evolution Strategy and Random Search).
For the case of radio networks and their KPI prediction, I suggest a co-existence of federated and cell-wise learning. Thus, each cell will contain both, a shared federated model which can be transferred to new cells, and a local model to predict future KPI values. Additionally, more complex hybrid models than this one should be experimented with (e.g. ensemble methods, a model selector). Given a higher amount of data it might be better to forecast not only from a cell's training experience, but from the federation's, as it was the case for the Google keyboard [MR17]. One particular hybrid model that I suggest is proposed by Hu, Zhang and Zhou [HZZ16]. Their approach (originally meant for Transfer Learning) if applied to federated learning would consist of performing the *federated averaging* on all neural network layers except the last one to learn the time series patterns from all federated cells. Then, the last layer would be only updated locally for learning the cells' individual behaviour.

All the same, considerations to be taken into account are KPI autocorrelation, presence of spikes, and clustering cells with highest similarity. These could be tackled in the following manner: using anomaly detection/prediction for discarding (from the federated training) cells with high number of spikes and unstable or low autocorrelation. Then, these problematic cells should be only trained individually.

I also suggest acquiring higher resolution (e.g. per minute, 500ms, or every 100ms) in data sampling to have more instances to learn from. The reason for this is that deep learning models typically learn better when the magnitude of data samples is in the order of millions, while traditional machine learning approaches tend to plateau

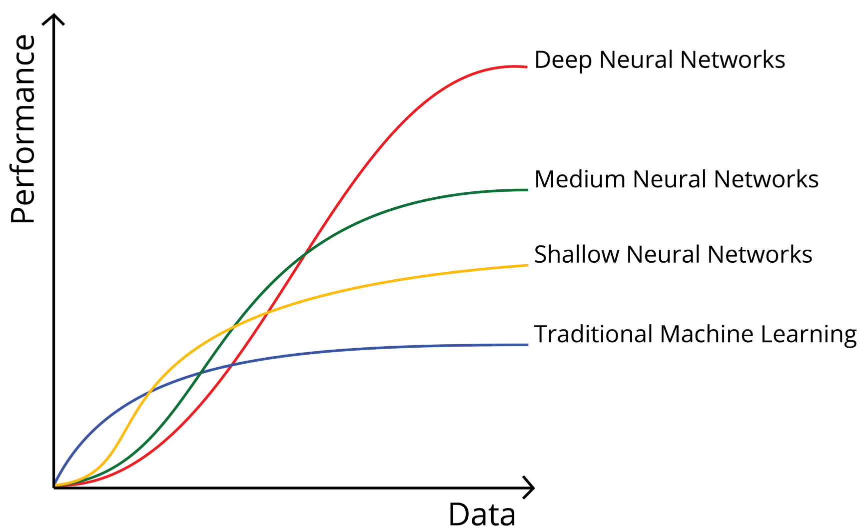



Figure 22: Data vs Performance. Image credits [Cor17]

after a certain amount of data [Ng18] (See Figure 22). However, it should be noted that hardware and software should be able to handle an increment of computation, since input and output sequences will also have to increase in order to keep the desired forecasts (at 15 minutes, 1 hour, and 6 hours).

Last but not least, I would suggest to use more adhoc libraries/functions such as tensorflow-federated [ten19] or the combination of Pytorch and PySyft [RTD+18].

# Acknowledgements

I am very grateful to Nokia for providing me the internship, domain expert guideline and the needed telecom data samples which allowed me to perform the experiments presented here. This thesis was also made possible thanks to the funding provided by Fondo para el Desarrollo de Recursos Humanos (FIDERH), which allowed me to pursue my masters degree at the University of Helsinki.
I would also like to thank my supervisors Sasu and Antti for their huge patience and support they gave me throughout my thesis writing. Last but not least, I would like to thank my family and my girlfriend Karina, for motivating me and believing in me when obstacles seemed too hard to overcome.

# References

Ago18 Agomuoh, F., AI-powered smartphones and the features that will make you want to buy them, 2018. `https://www.businessinsider.com/ai-smartphones-artificial-intelligence-features-phones-2018-1?r=US&IR=T`. [23.7.2019]

BEG+19 Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečný, J., Mazzocchi, S., McMahan, H. B., Overveldt, T. V., Petrou, D., Ramage, D. and Roselander, J., Towards federated learning at scale: system design. *SysML 2019*, 2019.

BJ70 Box, G. and Jenkins, G., *Time series analysis, forecasting and control.* Holden-Day, San Francisco, CA, 1970.

BYC13 Bergstra, J., Yamins, D. and Cox, D. D., Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML13. JMLR.org, 2013, pages 115–123.

Cau47 Cauchy, A., Méthode générale pour la résolution des systèmes d'équations simultanées. *Comp. Rend. Sci. Paris*, 1847, pages 536–538.

Cor17 Correa, A., [image] Building AI applications using deep learning, 2017. `https://blog.easysol.net/building-ai-applications/`. [18.09.2019]

Cyb89 Cybenko, G., Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2,4(1989), pages 303–314.

DCM+12 Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V. et al., Large scale distributed deep networks. *Advances in neural information processing systems*, 2012, pages 1223–1231.

Die15 Dietrich, D., *Data science and big data analytics : discovering, analyzing, visualizing and presenting data.* John Wiley & Sons, Indianpolis, IN, 2015.

FH19 Feurer, M. and Hutter, F. *Hyperparameter optimization*, pages 3–33. Springer International Publishing, Cham, 2019. URL https://doi.org/10.1007/978-3-030-05318-5_1.

Fra18 Frazier, P. I., A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

GB10 Glorot, X. and Bengio, Y., Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pages 249–256.

GB16 Guo, C. and Berkhahn, F., Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737*, 2016.

GBB11 Glorot, X., Bordes, A. and Bengio, Y., Deep sparse rectifier neural networks. *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pages 315–323.

GBC16 Goodfellow, I., Bengio, Y. and Courville, A., *Deep learning.* MIT Press, 2016. http://www.deeplearningbook.org.

GWHT13 Gareth, J., Witten, D., Hastie, T. and Tibshirani, R., *An introduction to statistical learning : with applications in R.* Springer, New York, NY, 2013.

Har18 Hartmann, F., Federated learning, 2018. https://florian.github.io/federated-learning/. [17.4.2019]

Hot33 Hotelling, H., Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24,6(1933), pages 417–441.

How19 Howard, J., Structured and time series data, Fastai, 2019. URL https://github.com/fastai/fastai/blob/master/courses/dl1/lesson3-rossman.ipynb.

HS97 Hochreiter, S. and Schmidhuber, J., Long short-term memory. *Neural computation*, 9,8(1997), pages 1735–1780.

HZZ16 Hu, Q., Zhang, R. and Zhou, Y., Transfer learning for short-term wind speed prediction with deep neural networks. *Renewable Energy*, 85, pages 83–95.

IS15 Ioffe, S. and Szegedy, C., Batch normalization: accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

KDH+19 Klein, A., Dai, Z., Hutter, F., Lawrence, N. and Gonzalez, J., Meta-surrogate benchmarking for hyperparameter optimization. *Advances in Neural Information Processing Systems*, 2019, pages 6267–6277.

KMY+16 Konečný, J., McMahan, H. B., Yu, F. X., Richtarik, P., Suresh, A. T. and Bacon, D., Federated learning: strategies for improving communication efficiency. *NIPS Workshop on Private Multi-Party Machine Learning*, 2016, URL `https://arxiv.org/abs/1610.05492`.

KP17 Krstanovic, S. and Paulheim, H., Ensembles of recurrent neural networks for robust time series forecasting. *International Conference on Innovative Techniques and Applications of Artificial Intelligence*. Springer, 2017, pages 34–46.

Las18 Lasse, K., State of the iot 2018: Number of iot devices now at 7b, 2018. IoT Analytics. `https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/`. [18.7.2019]

LBG+12 Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A. and Hellerstein, J. M., Distributed graphlab. *Proceedings of the VLDB Endowment*, 5,8(2012), pages 716–727. URL `https://arxiv.org/abs/1204.6078`.

McG17 McGuinness, K., [image] Deep learning workshop: transfer learning, 2017. URL `https://www.slideshare.net/xavigiro/transfer-learning-d2l4-insightdcu-machine-learning-workshop-2017`.

MHM10 McDonald, R., Hall, K. and Mann, G., Distributed training strategies for the structured perceptron. *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pages 456–464.

MHM18 McInnes, L., Healy, J. and Melville, J., Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Mit97 Mitchell, T., *Machine learning.* McGraw-Hill, New York, 1997.

MMR+16 McMahan, H. B., Moore, E., Ramage, D., Hampson, S. and y Arcas, B. A., Communication-efficient learning of deep networks from decentralized data, 2016.

MR17 McMahan, B. and Ramage, D., Federated learning: privacy-preserving collaborative machine learning without centralized training data, 2017. `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`. [18.09.2019]

MSA18 Makridakis, S., Spiliotis, E. and Assimakopoulos, V., Statistical and machine learning forecasting methods: concerns and ways forward. *PloS one*, 13,3(2018).

Mur12 Murphy, K., *Machine learning : a probabilistic perspective.* MIT Press, Cambridge, Massachusetts, 2012.

New18 Newzoo, Newzoo global mobile market report 2018, 2018. `https://newzoo.com/insights/trend-reports/newzoo-global-mobile-market-report-2018-light-version/`. [18.7.2019]

Ng18 Ng, A., *Machine learning yearning.* deeplearning.ai, 2018. `https://www.deeplearning.ai/machine-learning-yearning/`.

NH10 Nair, V. and Hinton, G. E., Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pages 807–814.

Nie19 Nielsen, A., *Practical time series analysis: prediction with statistics and machine learning.* O'Reilly Media, 2019. `https://learning.oreilly.com/library/view/practical-time-series/9781492041641/?ar`.

Ola15 Olah, C., [images] Understanding lstm networks, 2015. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. [25.6.2019]

Pal16 Palma, W., *Time series analysis.* John Wiley & Sons, 2016. ProQuest Ebook Central, `https://ebookcentral-proquest-com.libproxy.helsinki.fi/lib/helsinki-ebooks/detail.action?docID=4517503`.

Pea01 Pearson, K., On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2,11(1901), pages 559–572.

PZK14 Povey, D., Zhang, X. and Khudanpur, S., Parallel training of deep neural networks with natural gradient and parameter averaging. *arXiv preprint arXiv:1410.7455*, 2014.

RHW86 Rumelhart, D., Hinton, G. and Williams, R., Learning representations by back-propagating errors. *Nature*, 1986.

RTD+18 Ryffel, T., Trask, A., Dahl, M., Wagner, B., Mancuso, J., Rueckert, D. and Passerat-Palmbach, J., A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*, 2018.

SBS18 Sarkar, D., Bali, R. and Sharma, T., *Practical machine learning with Python.* Springer, 2018.

SCYE17 Sze, V., Chen, Y.-H., Yang, T.-J. and Emer, J. S., Efficient processing of deep neural networks: a tutorial and survey. *Proceedings of the IEEE*, 105,12(2017), pages 2295–2329.

SLA12 Snoek, J., Larochelle, H. and Adams, R. P., Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 2012, pages 2951–2959.

SS17 Shumway, R. and Stoffer, D., *Time series analysis and its applications.* Springer, New York, NY, 2017.

ten19 tensorflow.org, Federated learning, 2019. URL `https://www.tensorflow.org/federated/federated_learning`.

THHLB13 Thornton, C., Hutter, F., Hoos, H. H. and Leyton-Brown, K., Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pages 847–855.

TS10 Torrey, L. and Shavlik, J., [image] Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, IGI Global, 2010, pages 242–264.

via18 viasat, [image] Multi-layer perceptron, 2018. URL https://medium.com/coinmonks/the-artificial-neural-networks-handbook-part-1-f9ceb0e376b4.

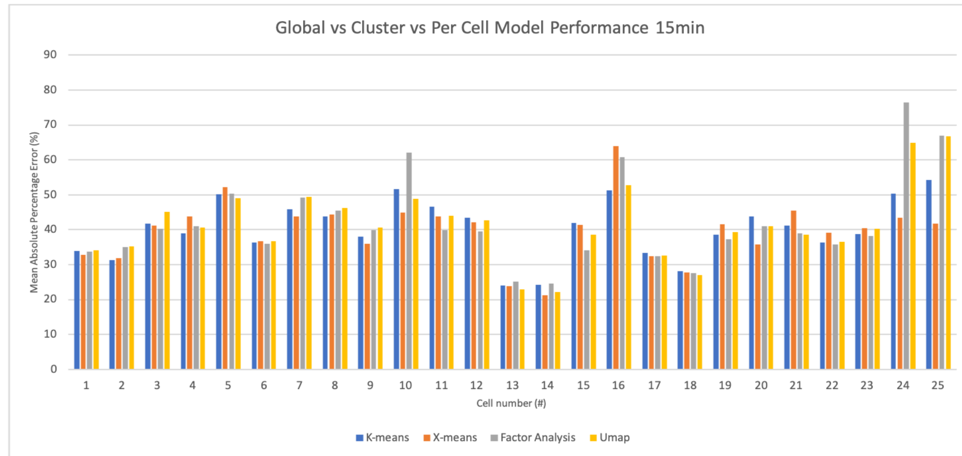
Whi51 Whittle, P., *Hypothesis testing in time series analysis*, volume 4. Almqvist & Wiksells boktr., 1951.

WM03 Wilson, D. R. and Martinez, T. R., The general inefficiency of batch training for gradient descent learning. *Neural networks*, 16,10(2003), pages 1429–1451.

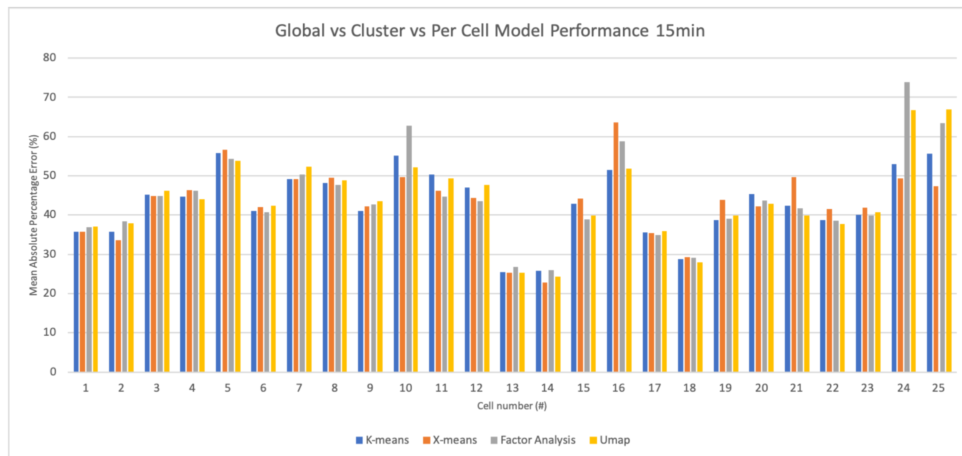
YM00 Yaffee, R. and McGee, M., *Introduction to time series analysis and forecasting*. Academic Press, New York, NY, 2000.

Zha18 Zhang, M., [lecture slides] Time series: autoregressive models, 2018. URL http://people.cs.pitt.edu/~milos/courses/cs3750/lectures/class16.pdf.

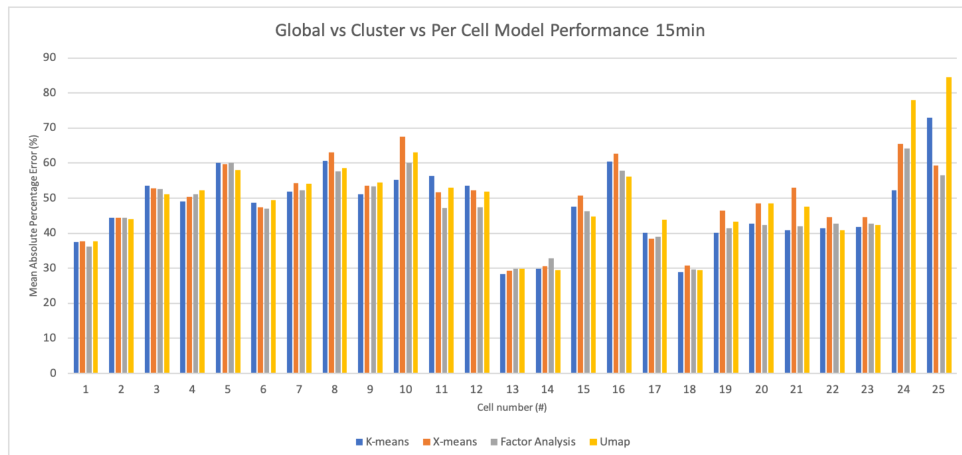
# Appendix A. Other Clustering Methods




(a) 15 minutes forecast




(b) 1 hour forecast




(c) 6 hours forecast

Figure 23: Clustering Techniques for Multicell forecast of target KPI