POLITECNICO DI TORINO

Master Degree in
Communications and Computer Networks Engineering

Master Thesis

A Deep Learning Analysis of Internet Traffic Datasets for 5G MEC Dimensioning

Supervisor
prof. Claudio Ettore Casetti

Candidate
Luigi Fabiano

Accademic Year 2019-2020

Abstract

This thesis work consists in studying Neural Network models in order to predict, as well as possible, the Internet traffic in a mobile network environment and use it in the 5G context for MEC (Multi-access Edge Computing) dimensioning.
In particular, the first chapter gives a brief introduction about AI and Machine Learning. Basic concepts needed to understand the mechanism behind Deep Learning are provided through the second chapter. Third chapter supplies a description of used datasets. Analysis and results of Neural Network models are presented in chapter four. Finally, the last chapter shows an application of the whole study for MEC dimensioning.

Contents

List of Figures

List of Tables

Chapter 1

AI and Machine Learning techniques

1.1 Artifical Intelligence

Artificial Intelligence (AI) is a wide computer science branch that aims to mimic the human behavior, abilities and intelligence in order to perform tasks in a reliable manner without or with the minimum effort. Nowadays, it is a very relevant component of the society. In fact, it is applied in a very large number of contexts providing different services and benefits: think about Home Automation Systems that improve and connect many typical technologies present in a house in order to make more comfortable the daily experiences; or to Virtual Assistants, capable of doing little and frequent tasks just by means of some commands; or more important and larger applications, designed mainly to improve safety, as the incoming Autonomous Vehicles.
All these different AI systems have a common factor: they are the result of learning process based on observations. This means that AI is able to build machines/systems devoted to a clearly specialized task by iteratively processing a large amount of data, obtained from observations, in order to find and recognize patterns and learn features from them. The amount of data needed to get a reliable application depends on its complexity: the more complex it is the larger must be the data from which has to be extracted the peculiar structure. This implies a certain level of required computational resources.
So, Artificial Intelligence is based on many methods and technologies able

to study the characteristic of human discipline and replicate its peculiar behaviors. The most used are Machine Learning algorithms.

1.2 Machine Learning

Machine Learning, subset of the Artificial Intelligence, is a statistical tool that defines how to learn from experience without being explicitly programmed. The learning process consists of observation of data, called training dataset, in order to find structure and regularities. From them a mathematical model is built. It is able to make its own conclusions, classification and predictions on new data, called test dataset.
The entire process is iterative. It can be described as a sequence of operations that are repeated until a certain performance level is reached: train the system, get results, collect feedbacks and use them to repeat the sequence. Obviously, the performance strongly depends on the problem complexity.

Machine Learning algorithms can be classified into:

- Supervised Learning: both input and output data are provided and labeled[1]. The built model, result of the learning process, is able to create a mapping function between input and output in such a way to make predictions on testing data. There are two main techniques defined by this category:
  - Classification: the model is a classifier. Given new input data, it assigns a discrete value which represents the corresponding predicted label. Image classification is the main example.
  - Regression: the trained model returns a continuous value, that is an estimation or a prediction, expressed in function of features values.

[1]Generally, with "Features" is denoted the input data, with "Label" or "Class" the output.

- Unsupervised Learning: only unlabeled input data is provided to the machine for the training process; as a consequence, a well defined output value is not obtained as in Supervised Learning but the model finds unknown patterns allowing to make classifications. In this way, they can be used to categorize input data. Clustering is a typical Unsupervised Learning technique that groups input according similarities basis.

- Reinforcement Learning: the algorithm focuses on making a sequence of decision to achieve a certain goal. It is based on trials and errors: actions are made starting from a random trial. For each action the system gets rewards or penalties. The object is to maximize the total reward. Several application fields exploit this kind of algorithm as AI gaming and real time decision.

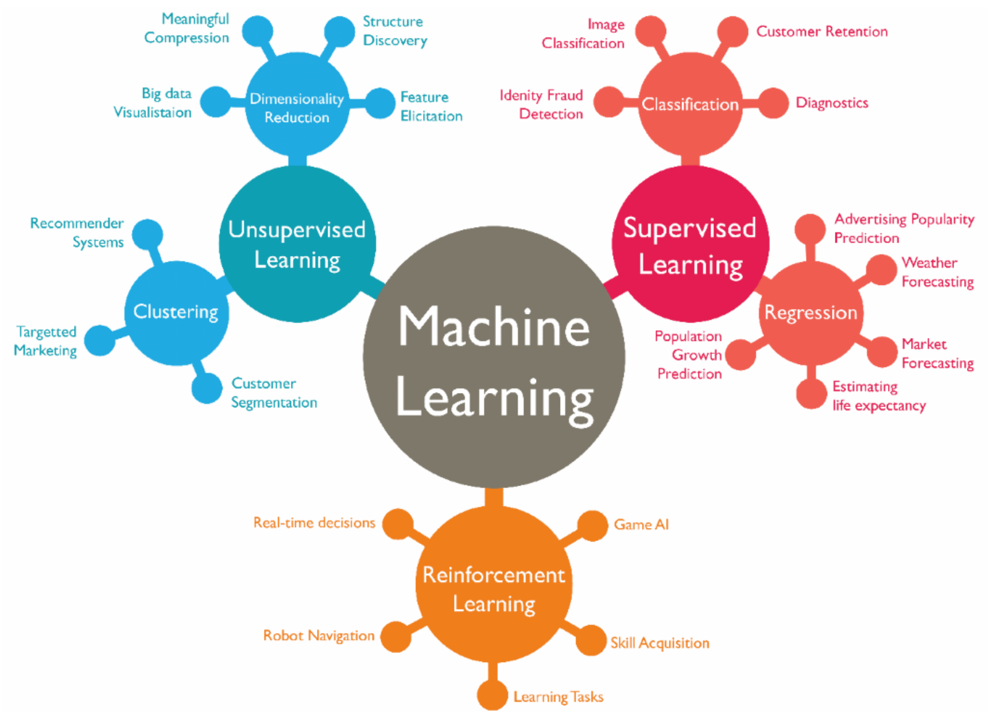


Figure 1.1: Machine Learning algorithms classification.
Figure taken from [1].

It is possible to combine Supervised and Unsupervised Learning to get the Semi-Supervised Learning: it uses a small amount of labeled data and a large amount of unlabeled data. In this way costs in labeling process are reduced. Generally, it presents improvement in learning accuracy.

So, the used algorithm depends the specific task problem. In this case, particular attention goes on Supervised Machine Learning algorithms.

1.2.1 Supervised Machine Learning

Supervised Machine Learning algorithms relate input and output in order to make classification or prediction. Several types of this algorithms have been defined along the time. Common algorithms are:

- Linear Regression: based on linear model, it computes target value through a linear combination of the input values.

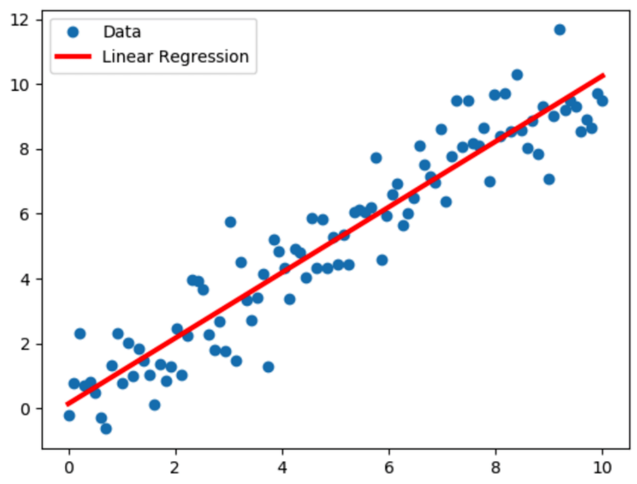


Figure 1.2: Linear regression model. Figure taken from [2].

- Decision Tree: it is based on a tree structure. Starting from the root node the input data is split according to its features values. The process is repeated for each created node (decision nodes) up to the leaf nodes that represent labels.

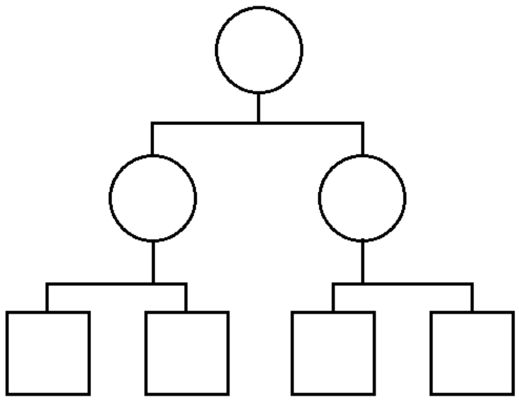


Figure 1.3: Decision tree model.

- Random Forest: considering an ensemble of decision trees, the final output value is the more occurred label among single trees in case of classification problem, the average among all the prediction in case of regression problem. This algorithm produces a more accurate output than single Decision Tree since models have a low correlation between them.

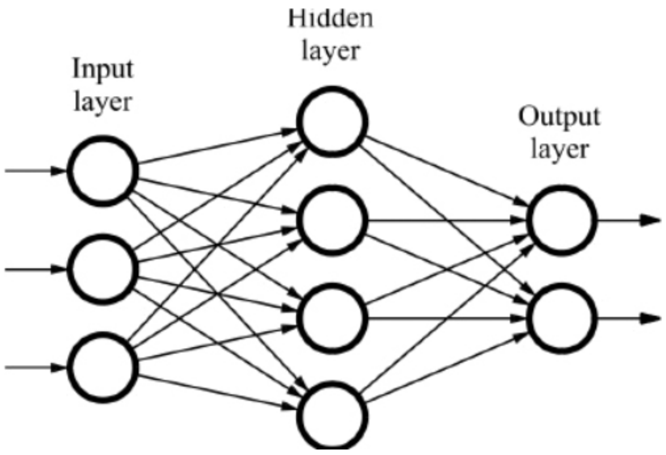


Figure 1.4: Neural Network model. Figure taken from [3].

- Neural Network: inspired by human brain model, is a network composed by nodes organized in layers (input, hidden and output layer).

Each node layer has a direct connection with the following nodes layer in order to forward the information elaborated on data according to a specific rule. The final layer produces a value (or values) that is either the predicted value is case of regression problem or the predicted label (or the probability to belong to each label of target data) in case of classification problem.

- Deep Leaning: it is based on Neural Network and is characterized by a higher number of hidden layers. It provides a very powerful solution able to find and learn very complex pattern in data.

Chapter 2

Deep Learning: analysis methodologies

2.1 Neural Networks and Deep Learning

Neural Networks (NN) are a particular type of Machine Learning algorithm that allows to define a model, given some data in input, in order to determine a final decision rule or a function to predict values.

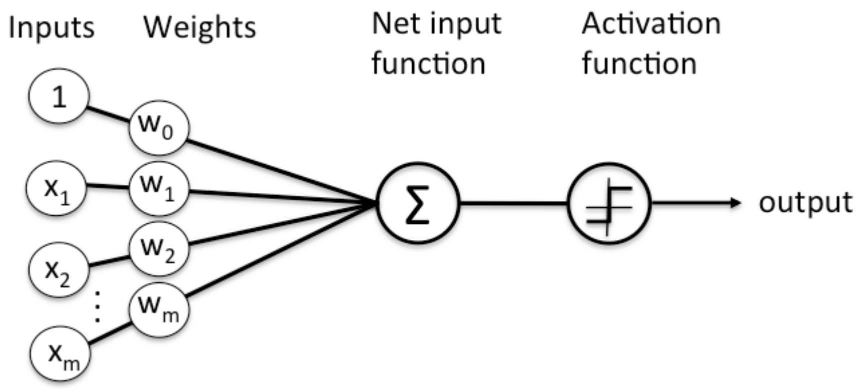


Figure 2.1: Neuron model. Figure taken from [5].

A Neural Network is composed by layers of nodes, called neurons, which are connected to all the adjcent layer nodes. Each neuron, leading actor, is responsible to combine the input features x_i with weights w_i in order to assign significance to them. To the sum $\sum_i w_i x_i$ is added a parameter b called bias. The result[2] $z = w \cdot x$, at this point, goes through the activation

[2]It is used the compact notation of the dot product $w \cdot x = \sum_i w_i x_i + b$, where w and x are vectors whose components are weights and input features.

function σ which is a non-linear complex function that produces an output signal[3] according to a rule $\sigma(z)$. This output becomes the input for each neuron in the next layer. This mechanism to feed as input the output of the previous layer, repeated up to the final one, defines the feed-forward property of the Neural Network.

Deep Learning is sub-field of Machine Learning based on Neural Networks. It exploits Deep Neural Networks that differs from the classic configuration for the presence of several layers between the input and output layer, called hidden layers.

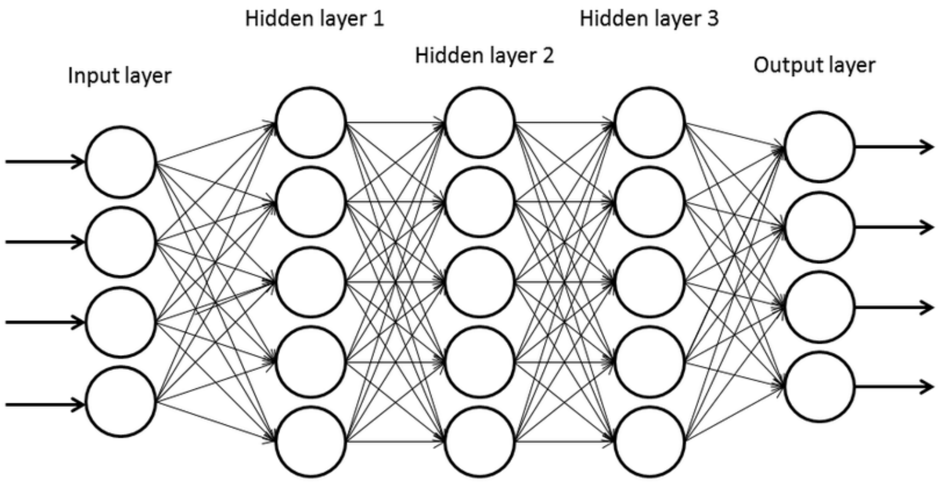


Figure 2.2: Deep Learning model. Figure taken from [6]

This system allow to discover complex structures in data that are not possible to find with a "simple" Neural Network. Hidden layers lead to another abstraction level: is like each one of them focuses on a particular feature of input data in order to recognize very complex pattern. Follows that the number of hidden layers is a parameter to define according to the task complexity: in general, a large number of hidden layers fit better a problem that presents very complex of data to be processed. Obviously, are required huge computational resources to guarantee high performance

[3]Whenever a neuron propoagates the signal producing an output different from 0 is defined activated.

levels.

2.1.1 Neural Network model

Given some input data x, composed by $[x_1, x_2, ..., x_i]$ features, and corresponding label $y(x)$, the Neural Network computes a model, through weights $w = [w_1, w_2, ..., w_i]$ and bias b, that it is able to calculate a prediction $\hat{y}$.
Learning process consists of finding optimal weights and biases[4] in such a way that the so called cost function is minimized. The cost function[5] $C(w, b)$ is a term, clearly dependent on learning parameters, that expresses the difference between the predicted value $\hat{y}$ and the original label value $y(x)$: the closer $\hat{y}$ is to $y(x)$ value the smaller is the loss; on the other hand, if $C(w, b)$ is large means that $\hat{y}$ is not a good approximation of $y(x)$.

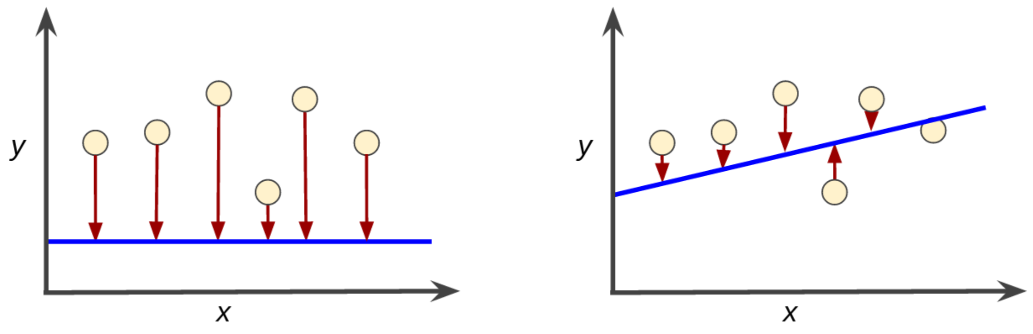


Figure 2.3: Cost function. Figure taken from [7].

One of the most used cost function is the quadratic cost function, also know as MSE (Mean Squared Error):

$$C(w, b) = \frac{1}{2n} \sum_x \|y(x) - \hat{y}\|^2 \qquad (2.1)$$

The technique used in learning process to train the system is called gradient descent: current values of weights and biases are updated in an

[4]Weights and biases are often referred as learning parameters.

[5]Cost function is also called loss or objective function.

iterative way with smaller ones in order to get the minimum value of the loss function. Actually, due to the not convexity of the cost function, there will be several local minimum values obtaining, generally, a sub-optimal solution to the minimization problem.

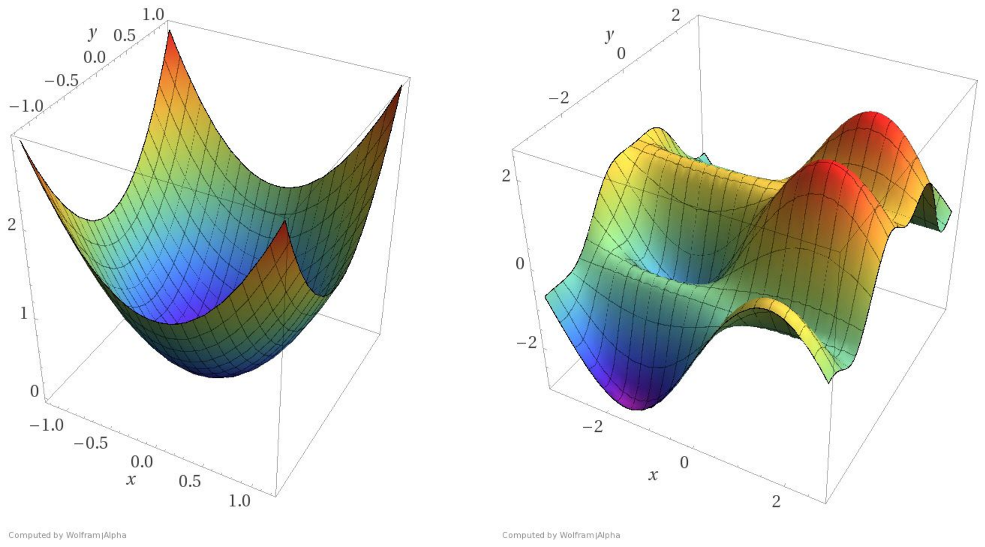


Figure 2.4: Ideal cost function vs. Real cost function.
Figure taken from [8].

Gradient descent updates weights and biases as follows:

$$w_k \rightarrow w_k^{'} = w_k - \eta \frac{\partial C}{\partial w_k} \tag{2.2}$$

$$b_l \rightarrow b_l^{'} = b_l - \eta \frac{\partial C}{\partial b_l} \tag{2.3}$$

where η is a parameter called learning rate that must be small enough otherwise it may lead to an increment of the cost function value. This algorithm works computing the gradient of the cost function[6], from which the derivatives with respect to weights and biases can be obtained for each input in order to evaluate correctly the cost function through their average.

[6]The gradient of the cost function is computed through the backpropagation algorithm.

So, it requires a lot of computation with a consequent slow down of learning process, particularly for large input data. In order to overcome this problem it is introduced the stochastic gradient descent which computes derivatives for small groups of m randomly chosen input, called batches. In this way learning parameters are updated as follows:

$$w_k \rightarrow w_k^{'} = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k} \tag{2.4}$$

$$b_l \rightarrow b_l^{'} = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l} \tag{2.5}$$

where X_j are the random inputs belonging to the batch j. This technique allows to obtain a speedup factor in the estimation of the gradient equal to $\frac{n}{m}$. Equations in this section are taken from [4].

2.1.2 Overfitting problem

Stressed many times, learning process is based on the analysis of input data from which weights and biases are computed in order to build a model. Generally, the amount of input data has to be sufficiently large in order to avoid what is called underfitting problem: Neural Network is not able to learn enough patterns to create a model to make predictions.

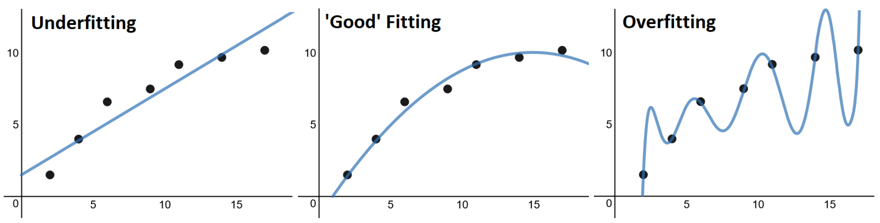


Figure 2.5: Fitting problems.
Figure taken from [9].

On the other hand, even if a large number of input data samples is provided, it could be small with respect the dimension of the Neural Network and, after an initial phase, the generalization of learning process stops adapting weights and biases on input data. Consequently, the obtained

model is not able to make correct prediction on test data. This describe the overfitting problem. It is a very big common problem in Neural Networks, particularly for Deep Neural Networks which have a very large number of hidden layers and neurons that means a lot of weights and biases.
One possible solution to this problem is to increase the number of input data samples. Actually, it does not provide a practicable solution since a further acquisition could be expensive or difficult to get. Thus, are introduced some techniques to help against overfitting allowing to have fixed input data:

- Regularization: the cost function is modified adding a penalty term, called regularization term. In particular, there are:
  - L2 regularization: it adds the sum of the squares of weights in the network

    $$C(w,b) = \frac{1}{2n}\sum_x \|y(x) - \hat{y}\|^2 + \frac{\lambda}{2n}\sum_w w^2 = C_0 + \frac{\lambda}{2n}\sum_w w^2 \quad (2.6)$$

    where C_0 denotes the original and unregularized cost function, λ (greater than 0) is the so-called regularization parameter and n is the input data size. Follows that weights are updated according to the formula:

    $$w \to w^{'} = \left(1 - \frac{\eta\lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w} \quad (2.7)$$

    When λ is large enough, weights are updated with smaller ones[7], iteration after iteration; when the regularization parameter is small the minimization of the cost function C_0 prevails. Small weights allow to generalize better the model response to pattern that are seen often in input data. A large λ means also large weights which may cause considerable variation of the network output even in

[7]The continuous smaller updating weights explain why L2 regularization is also called weight decay and $1 - \frac{\lambda}{2n}$ is known as weight decay factor.

case of small changes in input. Large weights are also obtained when a decrease in the unregularized cost function occurs avoiding to weights to go to 0. So, the regularization parameter λ synthesizes a compromise between finding small weights and minimizing the original cost function.
In case of stochastic gradient descent, L2 regularization weights updating becomes:

$$w \rightarrow w^{'} = \left(1 - \frac{\eta\lambda}{n}\right) w - \frac{\eta}{m} \sum_{x} \frac{\partial C_x}{\partial w} \tag{2.8}$$

– L1 regularization: penalty term is the sum of the absolute value of the weights. The regularized cost function is:

$$C(w,b) = \frac{1}{2n} \sum_{x} \|y(x) - \hat{y}\|^2 + \frac{\lambda}{n} \sum_{w} |w| = C_0 + \frac{\lambda}{n} \sum_{w} |w| \tag{2.9}$$

The resulting update rule is:

$$w \rightarrow w^{'} = w - \frac{\lambda}{n} sign(w) - \eta \frac{\partial C_0}{\partial} \tag{2.10}$$

For large weights it does not have a strong impact on updating operation. Instead, when weight values are small, regularization effects are significant, more than the ones provided by L2. So L1 regularization produces a concentration of weights a small number of high-importance connections, while the other weights tend to zero.

Biases terms are not involved in regularization since having large values does not imply high neuron sensitivity to its inputs as well as having large weights.

- Dropout: it is a particular regularization technique that acts directly on the Neural Network structure. For each batch learning session, a random subset of hidden layers neurons is deleted in order to update weights and biases through the "new" network. At this point, dropped neurons are restored and the process is repeated picking up another random group of neurons.

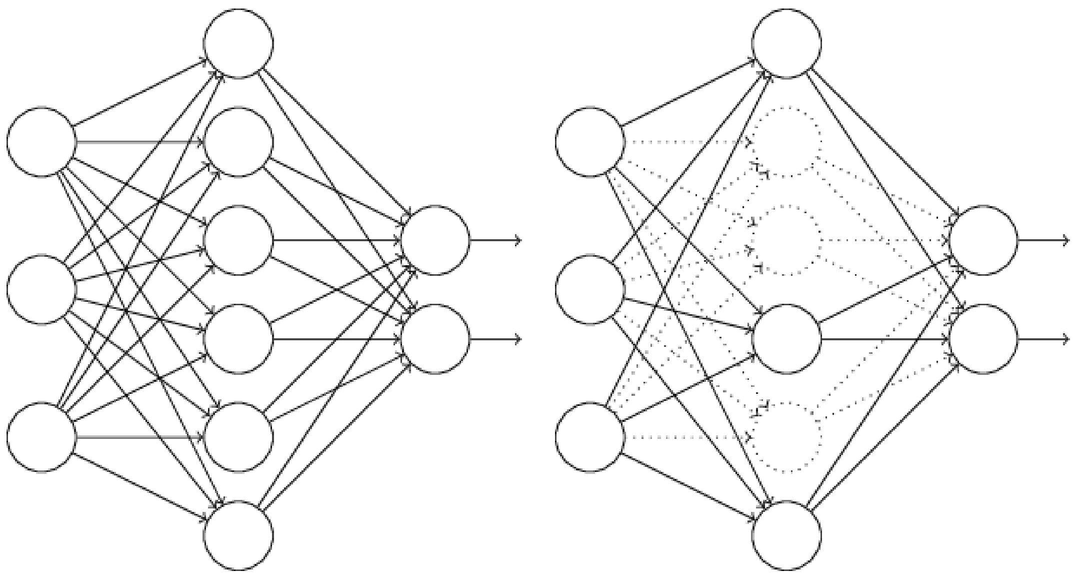

(a) Before dropout application. (b) After dropout application.

Figure 2.6: Dropout technique.
Figures taken from [10].

This procedure has the effect of training different Neural Networks and averages their results. Since different networks will overfit in different ways, the total Neural Network will be affected by reduced overfitting.

2.2 Neural Network implementation

Neural Network and Deep Neural Network implementation goes through different definitions and settings: datasets, hyper-parameters, activation function and regularization parameters.

2.2.1 Datasets

Different groups of data are required for a correct implementation of Neural Networks:

- Training dataset: it is composed by samples of data which are used for learning process. Neural Networks identify patterns and learns from them, updating weights and biases.

- Validation dataset: group of data used to provide an evaluation of the model during the learning process. So, it allows to tune correctly hyper-parameters[8] and detect overfitting problems.
- Test dataset: data given in input to the built model on which it is intended to get a classification or a prediction.

2.2.2 Hyper-parameters

Hyper-parameters have an enormous impact on learning process: the produced model and its predictive capacity strongly depends on them. They are the following:

- Batch size: number of training samples used in one single iteration of training process. In general, the smaller is the batch size value the better is the accuracy obtained for the model requiring a larger number of updates, that means larger computations and slower learning process.
- Epoch: number of steps through the entire training dataset during the training session. Good model performance is achieved exploiting several epochs.
- Learning rate: formally denoted with η, it controls the speed at which Neural Network learning parameters are updated. If it is too small, weights and biases move slowly in direction of the optimal values for cost function minimization. Moreover, the probability to get stuck in a local minima, without improvement during learning process, becomes higher. If it is too large the process could be unstable, overshooting weights and bias optimal value, and the algorithm might not converge.

[8]Hyperparameters are described in section 2.2.2.

- Hidden layers number.
- Number of neurons per layer.

In all hyper-parameters there is an intrinsic trade-off between performance level and computational costs: high accuracy values require ideal settings which involve very high number of computations, as learning parameters updating or cost function gradient evaluation, with consequent larger time and resources needs.

2.2.3 Activation function

The activation function is fundamental component in Neural Networks. It is a non-linear function that allows to extract information on high-dimensional and non-linear dataset in order to create complex mappings between inputs and outputs of the network.

For this thesis the following activation functions are used:

- Linear: the output signal is proportional to the neuron input

$$\sigma(z) = z$$

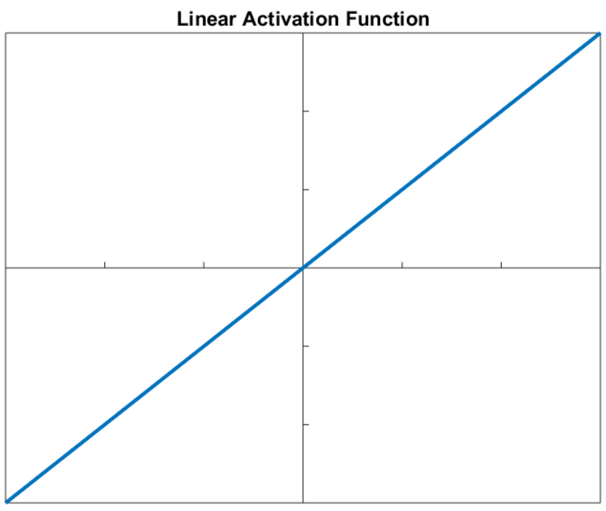


Figure 2.7: Linear activation function.

Since the derivative is constant, the gradient does not depend on the input value and both weights and biases are updated with the same updating factor. In this way, Neural Network will not understand which weights allow to improve the prediction.

It is usually employed as output layer activation function for regression problems.

- ReLU (Rectified Linear Unit): neurons are activated only if its input is greater than 0

$$\sigma(z) = \max(0, z)$$

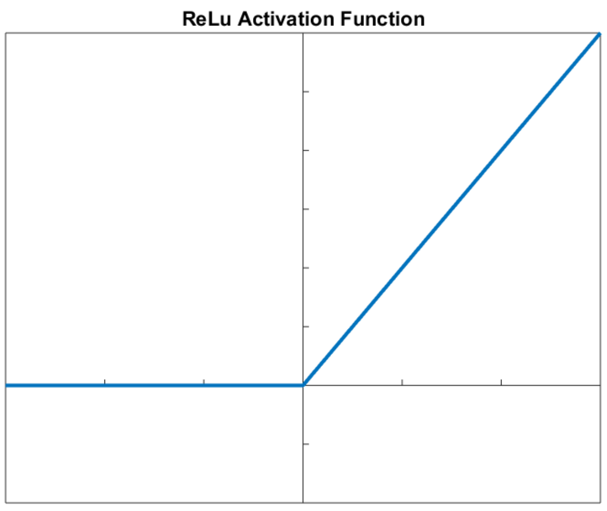


Figure 2.8: ReLU activation function.

Therefore, ReLU activation function does not activate all the neurons at the same time, providing a better computational efficiency and a faster convergence than other activation functions. By contrast, for input values smaller or equal to 0 the gradient is zero. This implies that some neurons can never be activated since their learning parameters are not updated.

It is commonly used in modern Neural Networks.

- Leaky ReLU: variation of ReLU activation function in which is present a linear component also for negative input values

$$\sigma(z) = \max(\alpha \cdot z, z)$$

  This modification allows to avoid dead neurons since gradient of input is always different from 0.

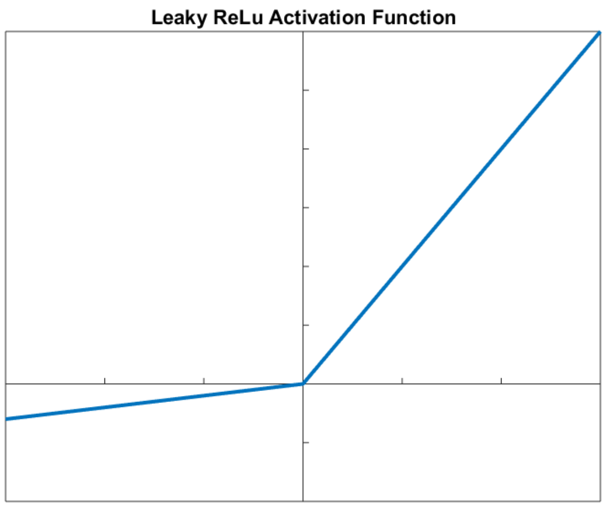


Figure 2.9: Leaky ReLU activation function.

- ELU (Exponential Linear Unit): differently from Leaky ReLU, this activation function relates negative inputs and the output of a neuron through a logarithmic curve

$$\sigma(z) = \begin{cases} z & z \geq 0 \\ \alpha(e^z - 1) & z < 0 \end{cases}$$

- Swish: the ouput of a neurons is obtained through

$$\sigma(z) = \frac{z}{1 + e^{-z}}$$

  Since it is a not monotonic function, output values may decrease even when the input values are increasing.

Swish has very computational efficiency properties showing better performance than ReLU on Deep Neural Network.

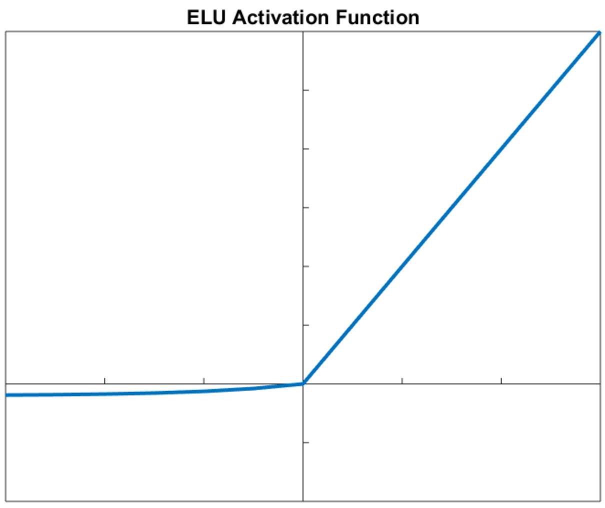


Figure 2.10: ELU activation function.

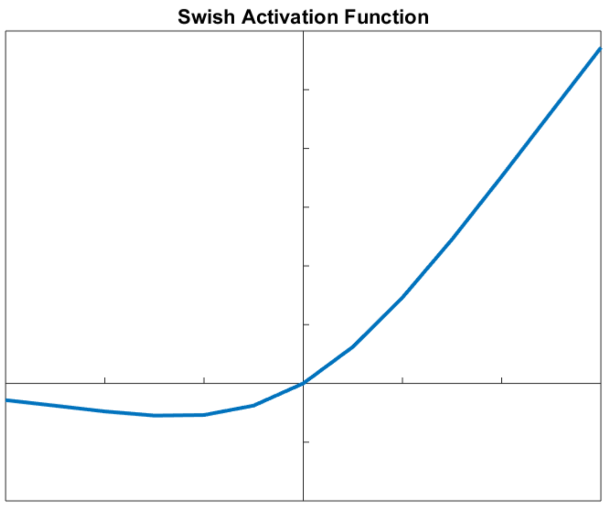


Figure 2.11: Swish activation function.

2.2.4 Regularization parameters

In Neural Networks, particularly in Deep Neural Networks, overfitting problem often occurs. In order to detect its action it is suggested to trace the cost function, or loss, on validation dataset.
As shown in section 2.1.2, some preventive techniques can be applied to reduce its effect. This implies that some terms must be defined:

- Regularization: the parameter λ must be defined. In particular, for L2 regularization, if it is too small the penality term has no effect, and unregularized cost function is minimized. If it is too large, weights are driven toward 0 with consequent underfitting problem.

- Dropout: the only parameter to be defined is the dropout rate: it is a number between 0 and 1 that represents the fraction of "deleted" neurons during the learning process.

Chapter 3

Dataset

3.1 Milano datasets

In order to study the Internet traffic through a Neural Network model, some dataset taken from [11] are exploited.

3.1.1 Grid dataset

Given Milano map it is defined a grid, through which the city and its suburbs is divided into 10.000 cells, each one of $250x250$ meters.

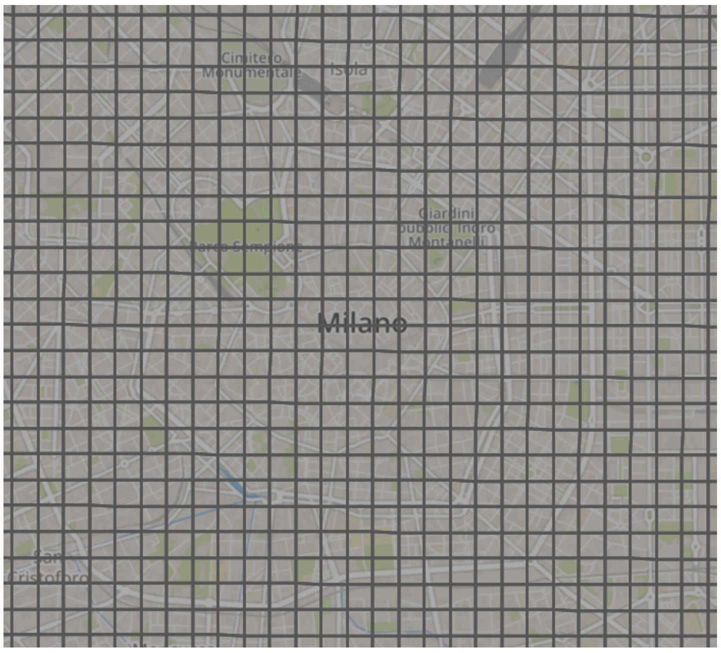


Figure 3.1: Milano grid.

The grid dataset, in .geoJSON format, gives information about the geographical coordinates that delimit each square[9].

3.1.2 Traffic dataset

Traffic dataset describes the information about the telecommunication activities over Milan city. It is composed by a set of files, one for each day, that contains the activities in November and December 2013. It is obtained from Call Detail Records (CDRs)[10] generated by Telecom Italia cellular network. In particular, CDRs are generated whenever:

- an SMS is sent or received.
- a call is generated or received.
- an Internet connection starts or ends and each time 15 minutes or 5 MB limits are reached from the last generated CDR.

Aggregating these information, the traffic dataset is defined according the following features:

- Square ID: it is represented by a number that is the identifier of the considered square of the Milan grid.
- Time Interval: this information is about the beginning of the time interval[11]. Each time interval has length of 10 minutes.
- Country Code: it is a country code of nation.
- SMS-in Activity: activity measurement of received SMS.

[9]The grid dataset will be useful in Chapter 5 in order to understand the traffic demand in a given area that will determine MECs position.

[10]CDRs are records generated by telecommunication companies for network management and building purpose: for any traffic activity of a mobile terminal it is recorded any entry with some information as IDs and BTS IDs.

[11]Time is provided in Unix Epoch format, which is the time elapsed, from January 1^{st}, 1970 at UTC, expressed in milliseconds.

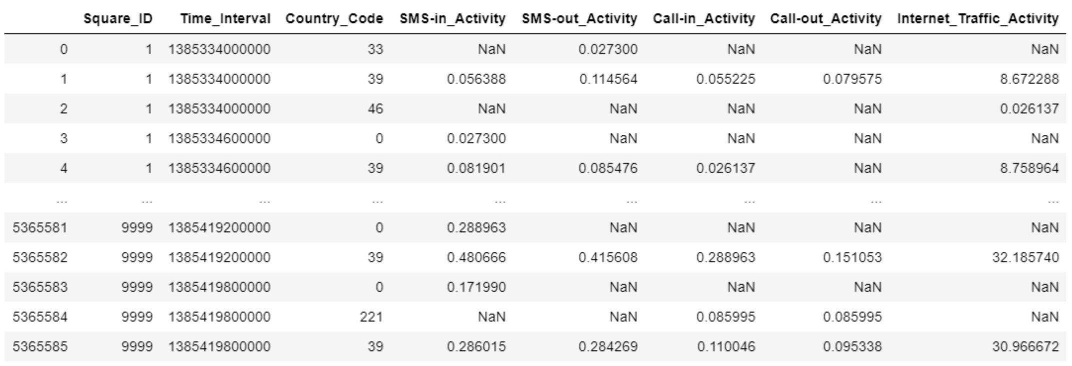
- SMS-out Activity: activity measurement of sent SMS.

- Call-in Activity: activity measurement of received calls.

- Call-out Activity: activity measurement of generated calls.

- Internet Traffic Activity: activity measurement of Internet traffic.

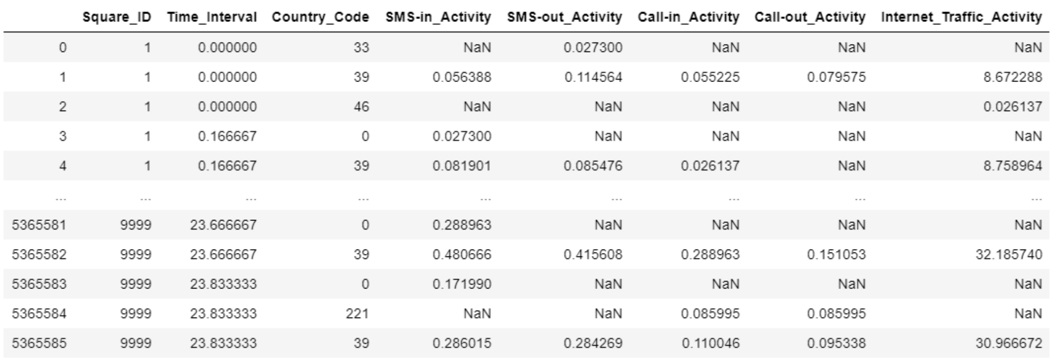
These fields are grouped according two policies which are: spatial aggregation, through which activity measurements are provided for each square Milano grid map, and temporal aggregation, through which activity measurements are grouped in 10 minutes time slots.



	Square_ID	Time_Interval	Country_Code	SMS-in_Activity	SMS-out_Activity	Call-in_Activity	Call-out_Activity	Internet_Traffic_Activity
0	1	1385334000000	33	NaN	0.027300	NaN	NaN	NaN
1	1	1385334000000	39	0.056388	0.114564	0.055225	0.079575	8.672288
2	1	1385334000000	46	NaN	NaN	NaN	NaN	0.026137
3	1	1385334600000	0	0.027300	NaN	NaN	NaN	NaN
4	1	1385334600000	39	0.081901	0.085476	0.026137	NaN	8.758964
...	...	...	...	...	...	...	...	...
5365581	9999	1385419200000	0	0.288963	NaN	NaN	NaN	NaN
5365582	9999	1385419200000	39	0.480666	0.415608	0.288963	0.151053	32.185740
5365583	9999	1385419800000	0	0.171990	NaN	NaN	NaN	NaN
5365584	9999	1385419800000	221	NaN	NaN	0.085995	0.085995	NaN
5365585	9999	1385419800000	39	0.286015	0.284269	0.110046	0.095338	30.966672

Figure 3.2: Traffic activity dataset - I.

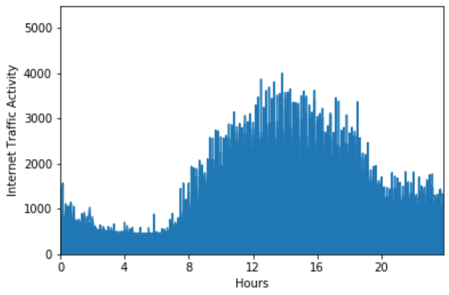
NaN values mean no activity was recorded for that specific field.
This traffic dataset, as will be shown in Chapter 4, is given as input to a Neural Network in order to build a model that is able to predict the Internet traffic demand. In order to allow a better generalization, it is modified expressing Time Interval field in terms of day hours.

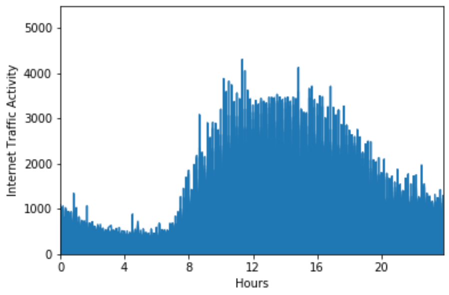


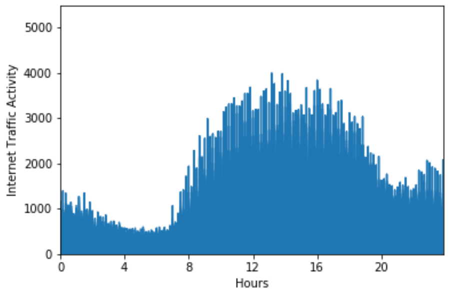
	Square_ID	Time_Interval	Country_Code	SMS-in_Activity	SMS-out_Activity	Call-in_Activity	Call-out_Activity	Internet_Traffic_Activity
0	1	0.000000	33	NaN	0.027300	NaN	NaN	NaN
1	1	0.000000	39	0.056388	0.114564	0.055225	0.079575	8.672288
2	1	0.000000	46	NaN	NaN	NaN	NaN	0.026137
3	1	0.166667	0	0.027300	NaN	NaN	NaN	NaN
4	1	0.166667	39	0.081901	0.085476	0.026137	NaN	8.758964
...	...	...	...	...	...	...	...	...
5365581	9999	23.666667	0	0.288963	NaN	NaN	NaN	NaN
5365582	9999	23.666667	39	0.480666	0.415608	0.288963	0.151053	32.185740
5365583	9999	23.833333	0	0.171990	NaN	NaN	NaN	NaN
5365584	9999	23.833333	221	NaN	NaN	0.085995	0.085995	NaN
5365585	9999	23.833333	39	0.286015	0.284269	0.110046	0.095338	30.966672

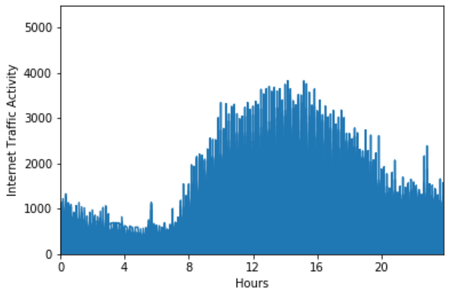
Figure 3.3: Traffic activity dataset - II.

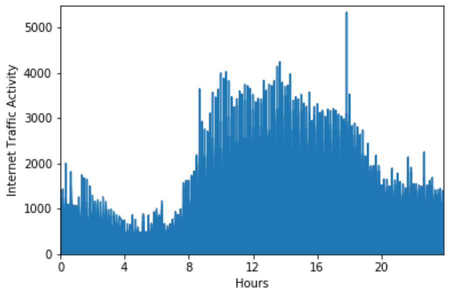
Internet Traffic Activity is the subject of the Neural Network study.

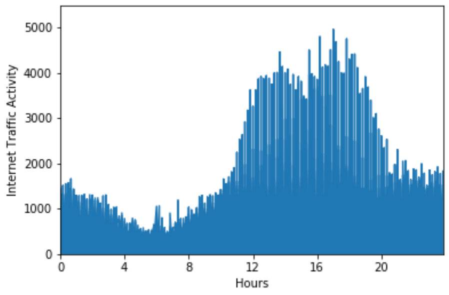

(a) Monday


(b) Tuesday


(c) Wednesday


(d) Thursday


(e) Friday


(f) Saturday

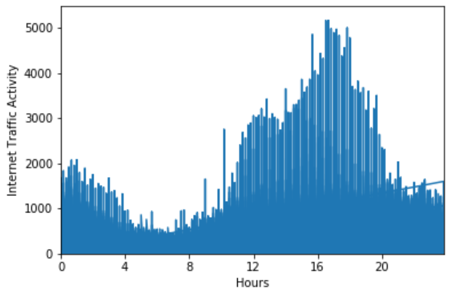
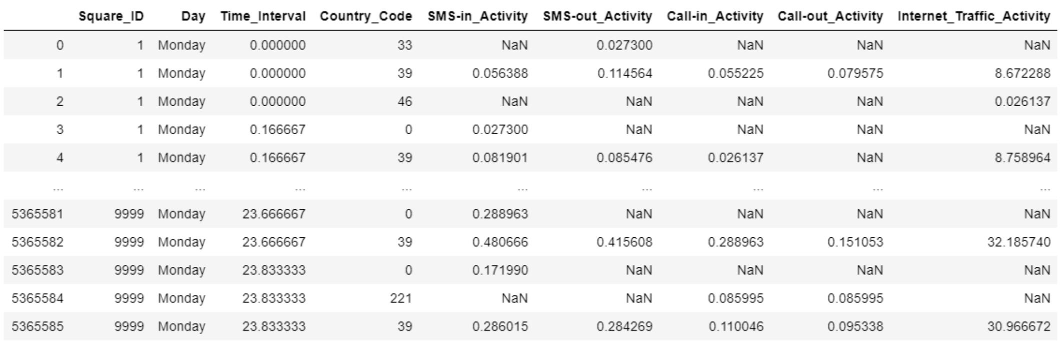

(g) Sunday

Figure 3.4: Internet traffic activity over days of the week

Despite just one week of Internet Traffic Activity is plotted, it presents its typical behavior. It is possible to notice how during the week, from Monday to Friday, the traffic activity seems to be regular, both in terms of time and quantity. Instead, the traffic trend is pretty different in Saturday and Sunday. For this reason, another modification on traffic dataset has been made inserting the day of the week in order to discriminate different days behavior.



	Square_ID	Day	Time_Interval	Country_Code	SMS-in_Activity	SMS-out_Activity	Call-in_Activity	Call-out_Activity	Internet_Traffic_Activity
0	1	Monday	0.000000	33	NaN	0.027300	NaN	NaN	NaN
1	1	Monday	0.000000	39	0.056388	0.114564	0.055225	0.079575	8.672288
2	1	Monday	0.000000	46	NaN	NaN	NaN	NaN	0.026137
3	1	Monday	0.166667	0	0.027300	NaN	NaN	NaN	NaN
4	1	Monday	0.166667	39	0.081901	0.085476	0.026137	NaN	8.758964
...	...	...	...	...	...	...	...	...	...
5365581	9999	Monday	23.666667	0	0.288963	NaN	NaN	NaN	NaN
5365582	9999	Monday	23.666667	39	0.480666	0.415608	0.288963	0.151053	32.185740
5365583	9999	Monday	23.833333	0	0.171990	NaN	NaN	NaN	NaN
5365584	9999	Monday	23.833333	221	NaN	NaN	0.085995	0.085995	NaN
5365585	9999	Monday	23.833333	39	0.286015	0.284269	0.110046	0.095338	30.966672

Figure 3.5: Traffic activity dataset - III.

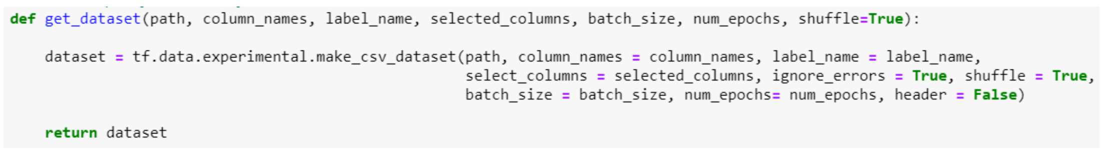
Chapter 4

Analysis and results

4.1 TensorFlow

Neural Networks in this chapter are implemented exclusively by means of Python language. In particular, it is exploited TensorFlow, an open source Machine Learning library that applies classes for applying ML algorithms to a lot of different data. It has a flexible environment of tools and libraries, like Keras, that allow to easily train and build Machine Learning models for very large number of applications through intuitive high-level API .

One of the most powerful functionalities provided by TensorFlow is data pipelines which enables to build complex input pipelines reading large amount of data from distributed files, that could even be of different formats, and performing transformation, as the division in batches or the shuffling operation.



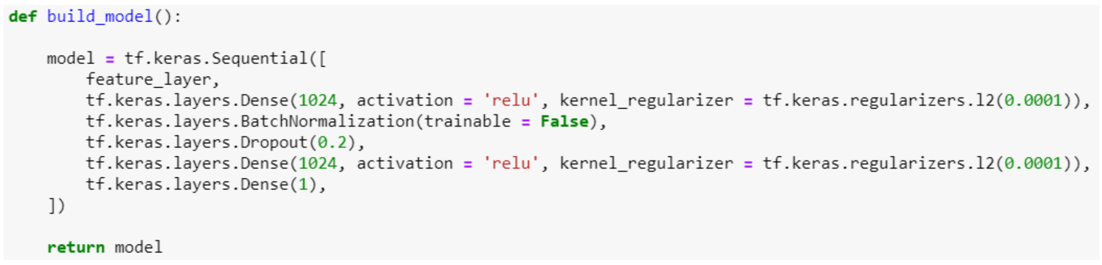
```
def get_dataset(path, column_names, label_name, selected_columns, batch_size, num_epochs, shuffle=True):

    dataset = tf.data.experimental.make_csv_dataset(path, column_names = column_names, label_name = label_name,
                                                    select_columns = selected_columns, ignore_errors = True, shuffle = True,
                                                    batch_size = batch_size, num_epochs= num_epochs, header = False)

    return dataset
```

Figure 4.1: TensorFlow: get dataset function.

TensorFlow allows to build simple Machine Learning algorithm, especially through Estimators API, as well as more complex Deep Learning models, through Keras that provides high-level API in order to build complex Deep Neural Networks.



```
def build_model():

    model = tf.keras.Sequential([
        feature_layer,
        tf.keras.layers.Dense(1024, activation = 'relu', kernel_regularizer = tf.keras.regularizers.l2(0.0001)),
        tf.keras.layers.BatchNormalization(trainable = False),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(1024, activation = 'relu', kernel_regularizer = tf.keras.regularizers.l2(0.0001)),
        tf.keras.layers.Dense(1),
    ])

    return model
```

Figure 4.2: TensorFlow: build model function.

4.2 Internet traffic model

This thesis work has the goal to build a model using Neural Networks in order to predict Internet traffic demand. In particular, it is done exploiting traffic dataset[12] (described in section 3.1.2) as follows: Square ID, Day[13], Time Interval, SMS-in Activity, SMS-out Activity, Call-in Activity, Call-out Activity are input features, used to predict Internet Traffic Activity label values.

Two cases are analyzed:

- No hidden layer analysis: using a Neural Network made up only of input and output layers, it focuses on hyper-parameters settings.
- One hidden Layer analysis: placing a hidden layer between input and output layer of the Neural Network, it looks for the best configuration.

Performances are evaluated in terms of validation loss, test loss and computational time. Validation loss is also used as reference to follow the model trend in order to tune correctly hyper-parameters.

4.3 No hidden layer analysis

The first step is to define datasets:

[12]All data are normalized in order to get a coherent model.

[13]The Day feature is introduces during the Dataset analysis.

- Training dataset: traffic data of the week November $4^{th} - 10^{th}$, 2013.
- Validation dataset: from Training dataset[14], traffic data of Sunday, November 10^{th}, 2013.
- Test dataset: traffic data of the week November $11^{th} - 17^{th}$, 2013.

Hyper-parameters (Section 2.2.2) determine the behavior of the Neural Network and the capacity of learn patterns from training dataset. In order to find the best performing model, are analyzed different settings:

- Batch size: [64, 128, 256, 512, 1024].
- Epochs: 10[15].
- Learning rate: [0.001, 0.01].
- Neurons:
  - Input layer: [64, 128, 256, 512, 1024].
  - Output ayer: 1.

Any combination of hyper-parameters is considered.

The number of learning parameters, that affects significantly computational complexity, is strictly related to input neurons number.

Neurons	Learning parameters
64	512
128	1,025
256	2,049
512	4,097
1024	8,193

Table 4.1: No hidden layer analysis learning parameters.

[14]Actually, it is obtained subtracting data to Training dataset. Practically, the Training dataset is composed by November $4^{th} - 09^{th}$, 2013 files.

[15]The number of epochs is imposed by computational limits.

4.3.1 Activation function analysis

Several activation functions are tested in order to understand which is the one that suits better the problem, improving Neural Network performance. In particular, are used ReLU, Leaky-ReLU, ELU and Swish activation functions.

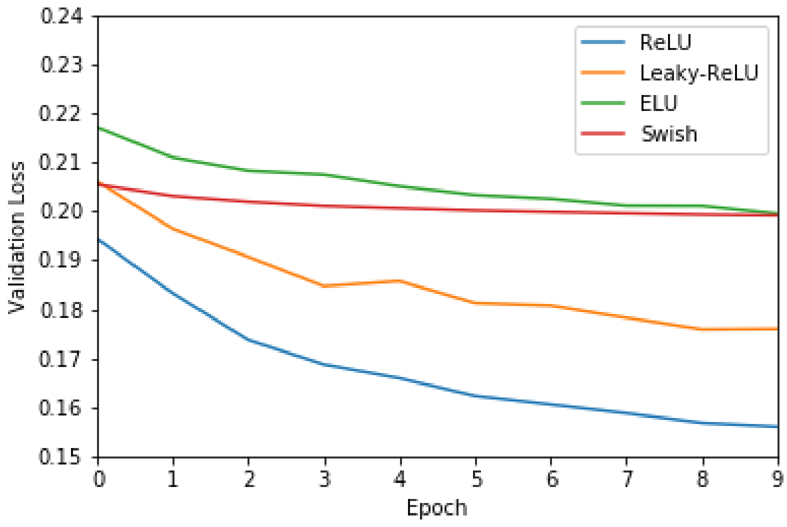
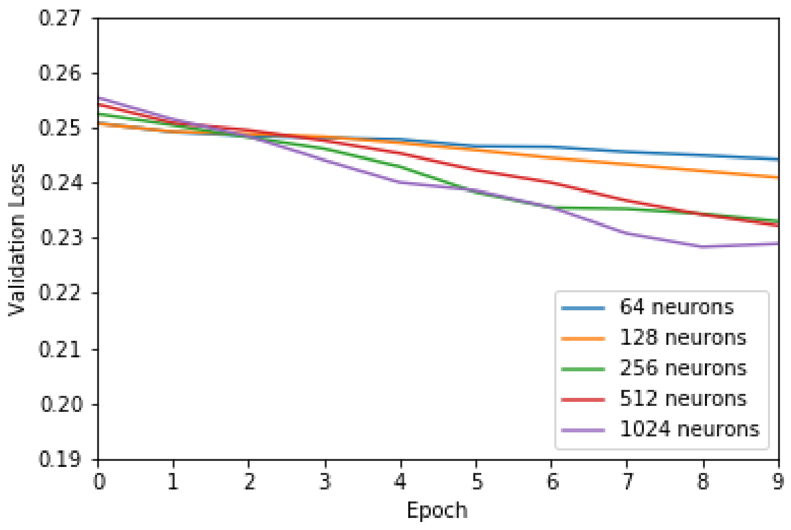


Figure 4.3: No hidden layer analysis validation loss: activation functions.

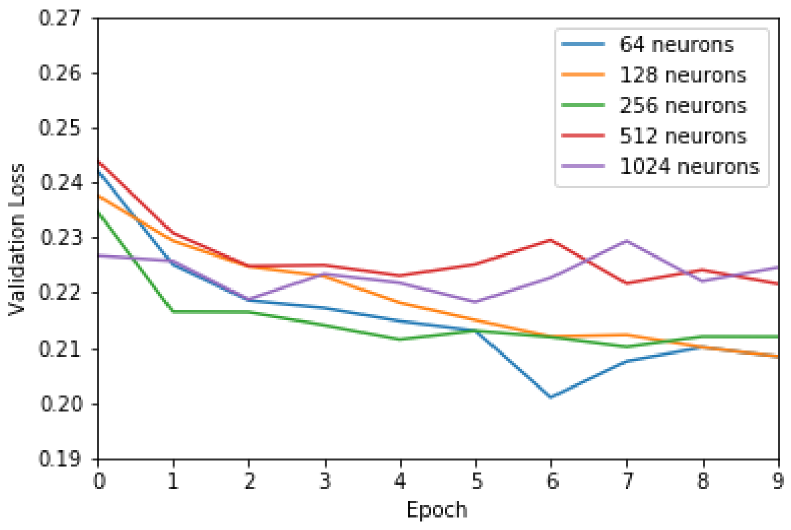
The plot shows the effect of each activation function: the ReLU seems to be the best one since it allows efficiently to minimize the cost function. For this reason, it is used for the rest of the analysis.

4.3.2 Batch size and learning rate analysis

Starting from batch size equal to 1024, both learning rate setting cases are examined obtaining the following results:



(a) Learning rate = 0.001



(b) Learning rate = 0.01

Figure 4.4: No hidden layer analysis validation loss: batch size = 1024.

- Learning rate = 0.001: from Figure 4.4a it is possible to notice how validation loss decreases as the number of epochs increases, for each input neurons number. Best performances in terms of Losses, according to Table 4.2a, are reached using the highest number of neurons. Instead, computational time increases with input layer size.

- Learning rate = 0.01: Figure 4.4b shows how validation loss trend starts to oscillate after some epochs for each neuron number. This suggests that learning rate 0.01 value is larger than the optimal one producing an unreliable model. Increasing loss values are gotten with respect to neurons number (Table 4.2b).

Neurons	Validation loss	Test loss	Time
64	0.2442	0.1844	53m 18s
128	0.2409	0.1824	55m 13s
256	0.2330	0.1738	56m 22s
512	0.2322	0.1741	58m 16s
1024	0.2289	0.1723	1h 9m 8s

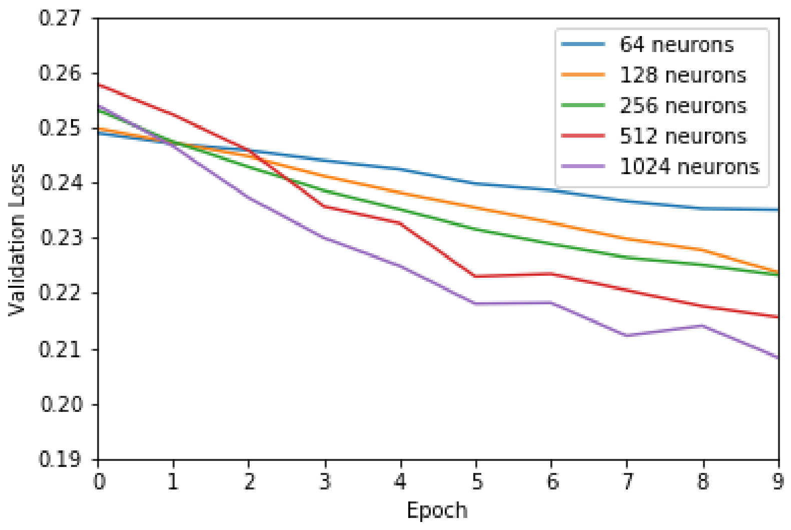
(a) Learning rate = 0.001

Neurons	Validation Loss	Test Loss	Time
64	0.2085	0.1550	47m 38s
128	0.2085	0.1596	49m 8s
256	0.2120	0.1582	53m 43s
512	0.2216	0.1767	56m 8s
1024	0.2246	0.1819	1h 10m 8s

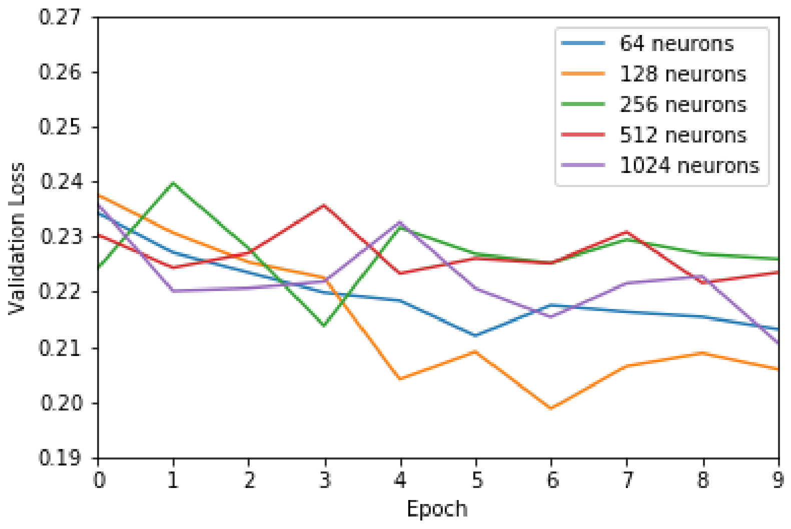
(b) Learning rate = 0.01

Table 4.2: No hidden layer analysis evaluations: batch size = 1024.

Setting batch size equal to 512, validation losses still decrease epoch after epoch with 0.001 learning rate value (Figure 4.5a) but faster: according to formulas 2.4 and 2.5 (Section 2.1.1), the speedup factor in the gradient estimation improves as batch size goes down. Consequently, are obtained better values in terms of losses. Instead, computational times become larger. Learning rate 0.01 is still too high.



(a) Learning rate = 0.001



(b) Learning rate = 0.01

Figure 4.5: No hidden layer analysis validation loss: batch size = 512.

Neurons	Validation loss	Test loss	Time
64	0.2351	0.1764	48m 52s
128	0.2238	0.1711	50m 21s
256	0.2233	0.1670	53m 47s
512	0.2156	0.1619	1h 2m 40s
1024	0.2084	0.1644	1h 16m 10s

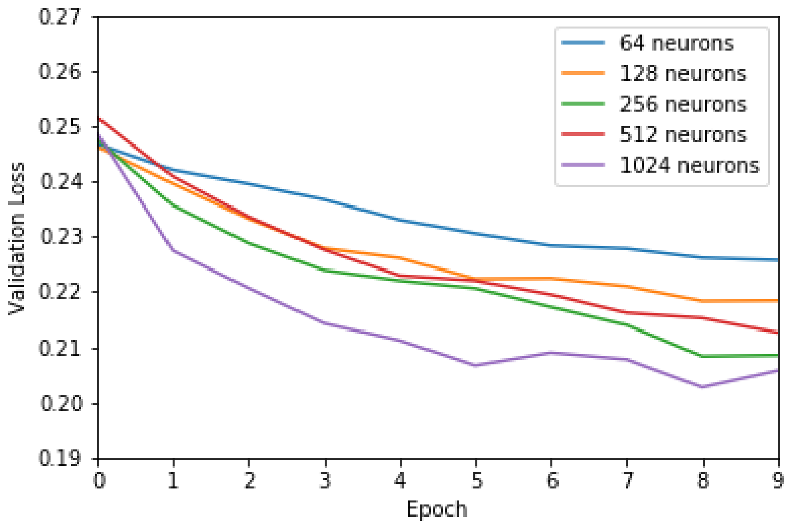
(a) Learning Rate = 0.001

Neurons	Validation Loss	Test Loss	Time
64	0.2132	0.1657	50m 54s
128	0.2060	0.1550	50m 51s
256	0.2260	0.1777	53m 10s
512	0.2235	0.1808	1h 3m 38s
1024	0.2108	0.1633	1h 12m 35s

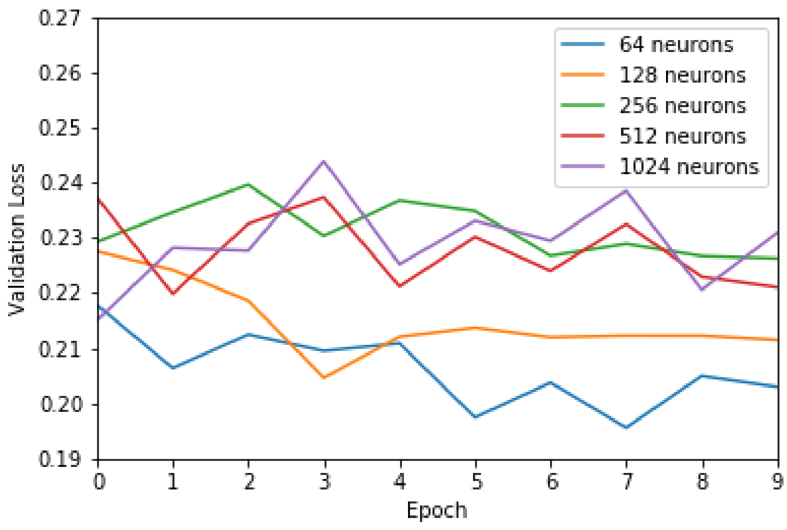
(b) Learning Rate = 0.01

Table 4.3: No hidden layer analysis evaluations: batch size = 512.

As expected, changing over and over the batch size it is consolidate the fact that smaller values allow to increase speedup factor in learning process getting better performances in terms of validation and test Losses for 0.001 learning rate, in spite of larger computation operations (larger learning parameters updates).



(a) Learning rate = 0.001



(b) Learning rate = 0.01

Figure 4.6: No hidden layer analysis validation loss: batch size = 256.

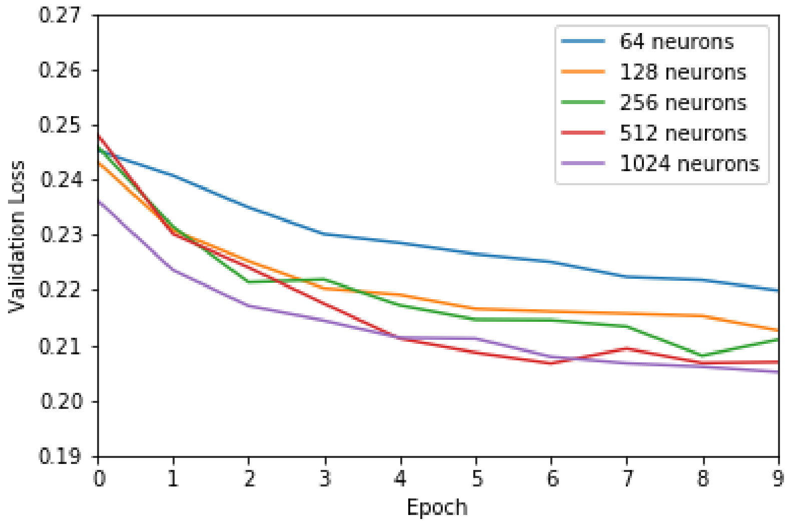
Neurons	Validation loss	Test loss	Time
64	0.2257	0.1689	55m 57s
128	0.2184	0.1636	56m 38s
256	0.2085	0.1587	57m 47s
512	0.2126	0.1721	1h 2m 31s
1024	0.2057	0.1591	1h 35m 26s

(a) Learning rate = 0.001

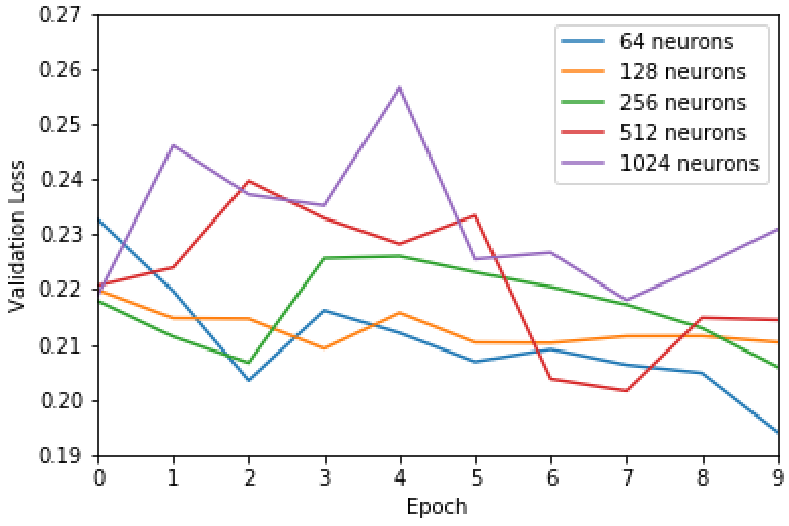
Neurons	Validation loss	Test loss	Time
64	0.2030	0.1499	55m 42s
128	0.2115	0.1603	55m 49s
256	0.2263	0.1823	56m 19s
512	0.2210	0.1782	1h 4m 21s
1024	0.2309	0.1870	1h 38m 30s

(b) Learning rate = 0.01

Table 4.4: No hidden layer analysis evaluations: batch size = 256.

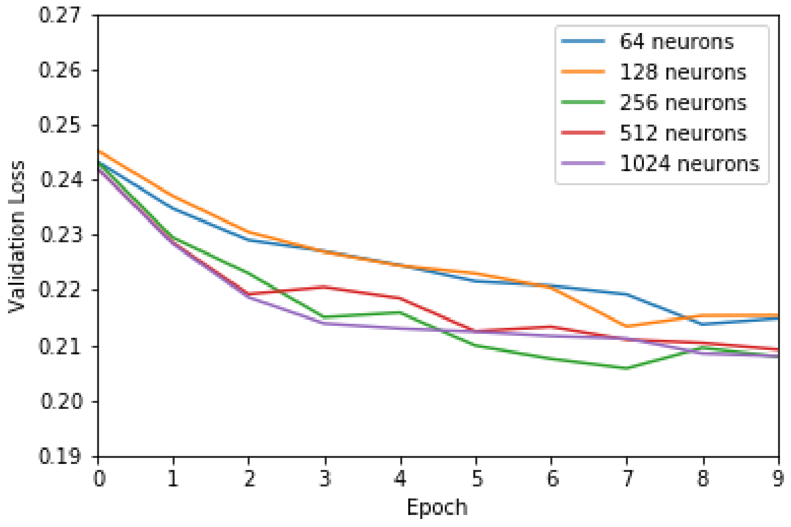


(a) Learning rate = 0.001

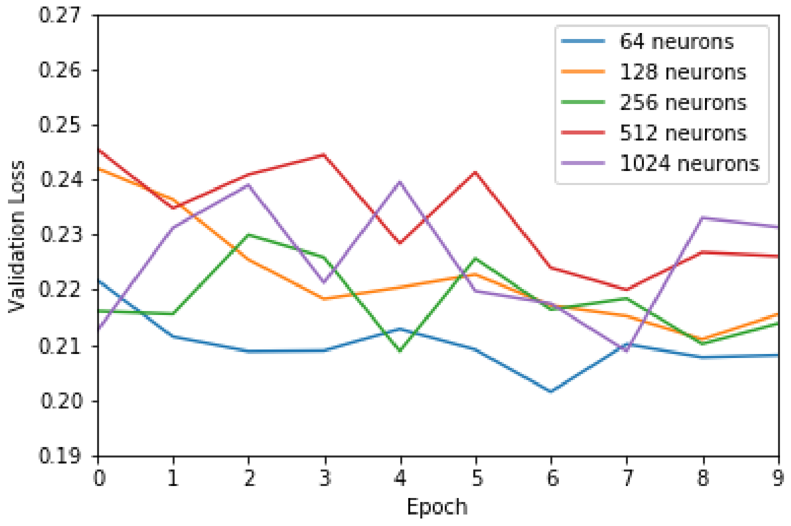


(b) Learning rate = 0.01

Figure 4.7: No hidden layer analysis validation loss: batch size = 128.



(a) Learning rate = 0.001



(b) Learning rate = 0.01

Figure 4.8: No hidden layer analysis validation loss: batch size = 64.

Neurons	Validation loss	Test loss	Time
64	0.2199	0.1651	1h 7m 33s
128	0.2127	0.1594	1h 5m 51s
256	0.2110	0.1576	1h 18m 27s
512	0.2070	0.1565	1h 25m 52s
1024	0.2052	0.1563	1h 44m 39s

(a) Learning rate = 0.001

Neurons	Validation loss	Test loss	Time
64	0.1942	0.1544	1h 14m 20s
128	0.2060	0.1574	1h 17m 6s
256	0.2104	0.1613	1h 11m 35s
512	0.2145	0.1722	1h 21m 57s
1024	0.2309	0.1868	1h 43m 34s

(b) Learning rate = 0.01

Table 4.5: No hidden layer analysis evaluations: batch size = 128.

Neurons	Validation Loss	Test Loss	Time
64	0.2149	0.1619	1h 56m 42s
128	0.2154	0.1619	1h 55m 51s
256	0.2080	0.1564	2h 2m 52s
512	0.2092	0.1580	2h 6m 41s
1024	0.2081	0.1581	2h 22m 18s

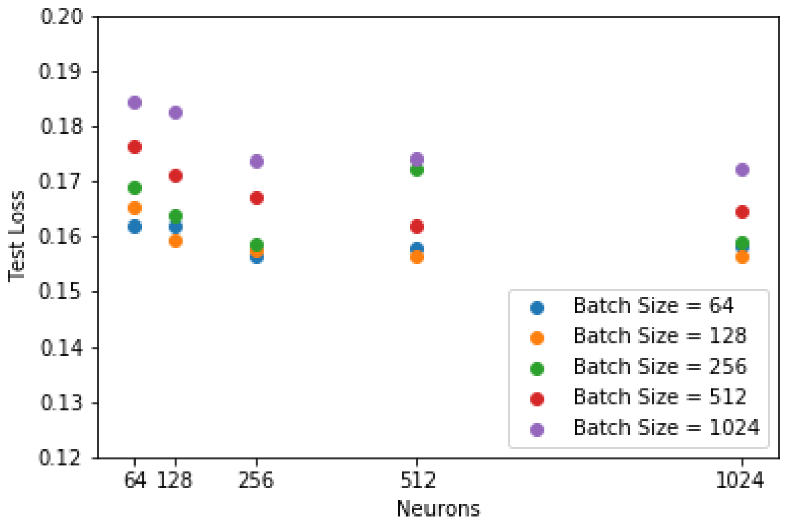
(a) Learning rate = 0.001

Neurons	Validation Loss	Test Loss	Time
64	0.2081	0.1550	1h 18m 40s
128	0.2155	0.1641	1h 56m 1s
256	0.2139	0.1657	2h 15m 6s
512	0.2261	0.1819	2h 8m 20s
1024	0.2314	0.1870	2h 27m 46s

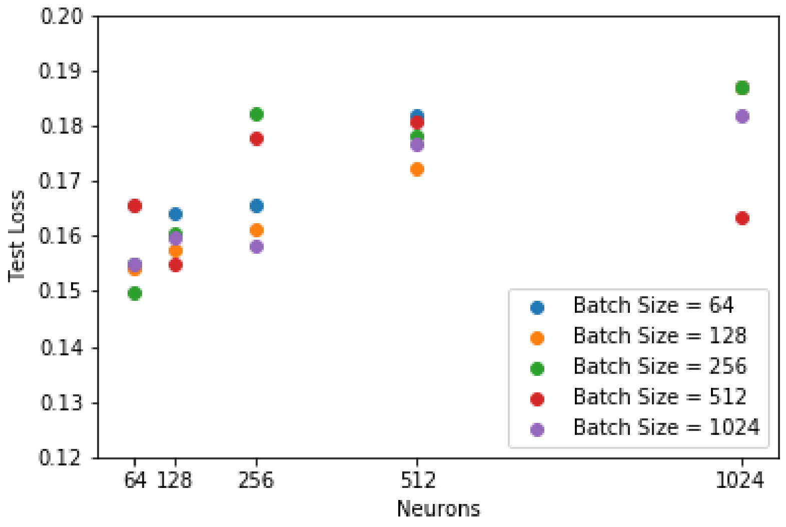
(b) Learning rate = 0.01

Table 4.6: No hidden layer analysis evaluations: batch size = 64.

Best performance models are mostly obtained using large input layer. As proof:



(a) Learning rate = 0.001



(b) Learning rate = 0.01

Figure 4.9: No hidden layer analysis test loss

Learning rate 0.01 setting always leads to an oscillating trend that can be defined random, proving that this value is too large and does not allow to reach minimum weights value. As a matter of fact, scatter plot shows the random behavior of computed models.

Running over 50 Epochs

Fixing the number of input neurons to 1024 and exploiting the "optimal" learning rate found (0.001), are computed models for each batch size over 50 epochs. Validation loss Figure 4.11 shows significant fluctuations also in this case. This is due to the large size of the Neural Network with respect to the input: overfitting problem starts to occur, with consequent generalization capacities loss. As a matter of fact, plotting the Training Loss and looking to its continuous decrease, it is evident that models continue to learn from Training dataset peculiarities.

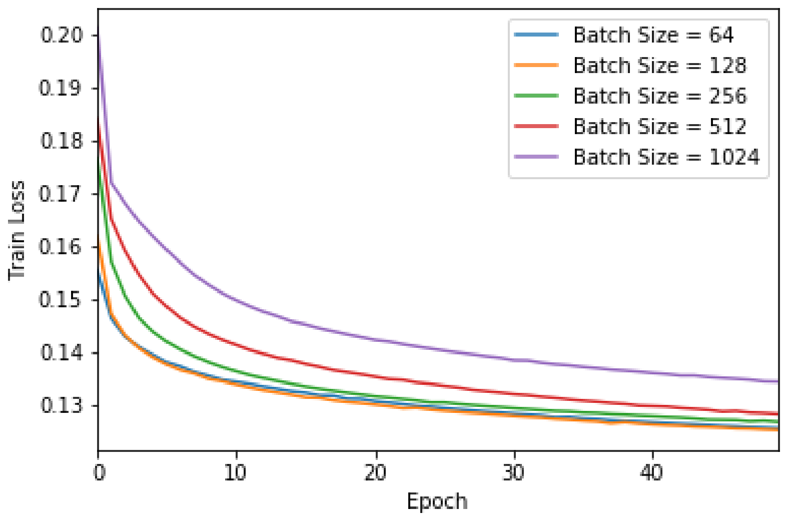


Figure 4.10: No hidden layer analysis train loss: epochs = 50.

Smaller batch sizes anticipate oscillating behavior with respect to the larger ones: this is due to their higher learning speedup factor that allows to reach sooner to the optimum achievable learning parameters configuration. Once again, from Table 4.7, losses improves while computational times get worse with increasing batch size.

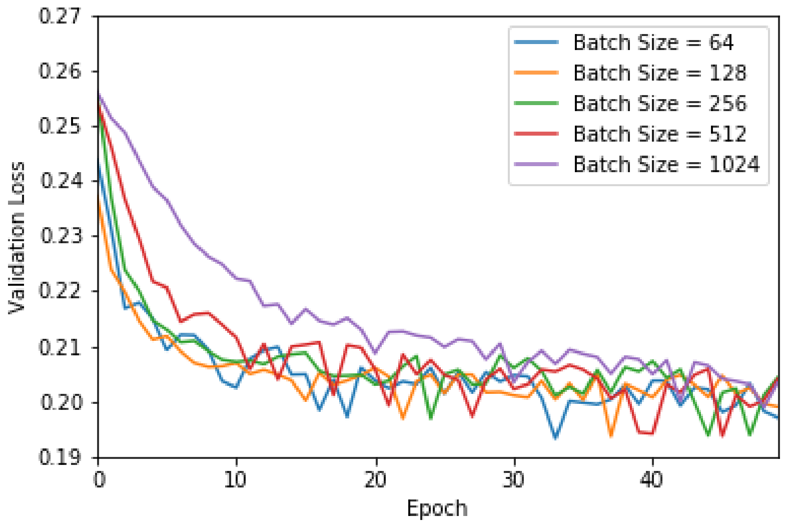


Figure 4.11: No hidden layer analysis validation loss: epochs = 50.

Batch size	Validation loss	Test loss	Time
1024	0.2043	0.1508	5h 30m 6s
512	0.2040	0.1424	5h 47m 34s
256	0.2034	0.1418	6h 13m 23s
128	0.1991	0.1410	8h 34m 13s
64	0.1971	0.1405	13h 49m 23s

Table 4.7: No hidden layer analysis validation loss: epochs = 50.

Validation and test losses comparison

Analyzing Figure 3.4 it is possible to notice how Internet traffic demand is different in terms of time and quantity between week-end days and the other days. In order to explain the significant gaps between validation losses and test losses, has to be specified that in all previous models validation loss is computed on Sunday, November 10^{th}, 2013 while test loss on the entire week November $11^{th} - 17^{th}$, 2013. So, it is easily understandable why built models produce a larger error on a single day of the week-end than the one computed over a complete week in which errors are compensated.

Therefore, the significant difference between the two values, for each case, is mostly determined by validation dataset choice. Indeed, changing the validation dataset, are obtained results in Figure 4.12. A further proof is given by Table 4.8 that shows all losses.

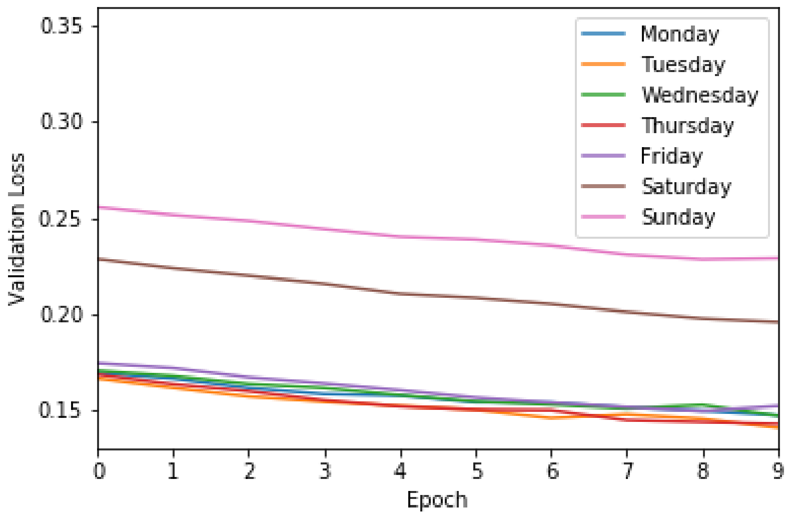


Figure 4.12: No hidden layer analysis validation loss: validation dataset choice.

Day	Validation loss	Test loss
Monday	0.1473	0.1633
Tuesday	0.1408	0.1615
Wednesday	0.1471	0.1633
Thursday	0.1428	0.1629
Friday	0.1520	0.1630
Saturday	0.1957	0.1625
Sunday	0.2289	0.1723

Table 4.8: No hidden layer analysis validation loss: validation dataset choice.

4.3.3 Datasets analysis

In order to improve performances, dataset are modified. In particular, are analyzed two cases:

- First case:
  - Training dataset: traffic data of two weeks, from November 4^{th} to November 17^{th}, 2013.
  - Validation dataset: from Training dataset, traffic data of Saturday, November 9^{th} and Wednesday, November 13^{th}, 2013.
  - Test dataset: traffic data of the week November $18^{th} - 24^{th}$, 2013.
- Second case:
  - Training dataset: traffic data of two weeks, from November 4^{th} to November 24^{th}, 2013.
  - Validation dataset: from Training Ddtaset, traffic data of Saturday, November 9^{th} and Wednesday, November 13^{th}, 2013.
  - Test dataset: traffic data of the week from November 25^{th} to December 1^{st}, 2013.

Validation dataset is composed of Saturday, November 9^{th} and Wednesday, November 13^{th} in both cases in order to get a more reliable values that approximates properly the test loss. Fixing the neuron number to 1024, learning rate equal to 0.001 and using the 1024 batch size, the following results are obtained:

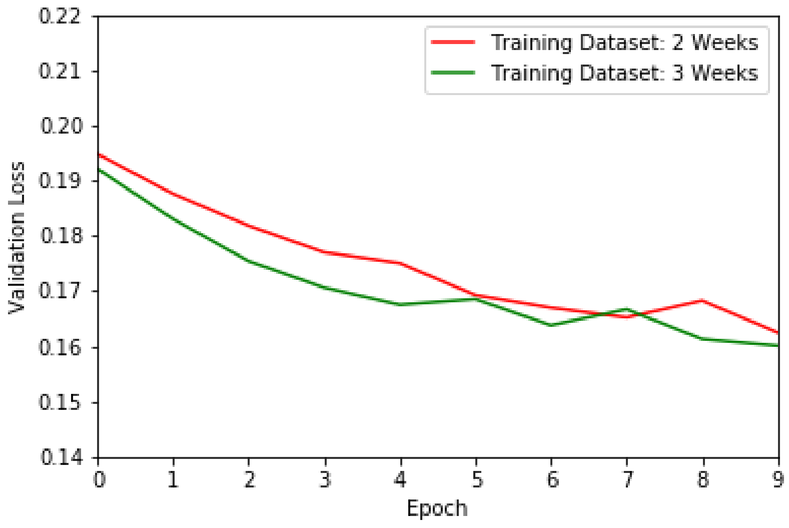


Figure 4.13: No hidden layer analysis validation loss:
training datasets comparison.

Training dataset	Validation loss	Test loss	Time
2 Weeks	0.1625	0.1581	2h 24m 46s
3 Weeks	0.1601	0.1610	3h 22m 11s

Table 4.9: No hidden layer analysis evaluations:
training datasets comparison.

Though losses values are very close, there are some differences. 2 weeks training dataset allows to get a smaller Test Loss. However, 3 weeks training dataset produces a better validation loss on the same validation dataset. This means it returns a better model and test losses differences could be explained by some unpredictable Internet traffic activities in the second case test dataset that determine a larger value.

At this point, according to the diversity of the traffic demand during the week, the idea is to split the training dataset. In particular, using 3 weeks training dataset and splitting it, are computed two different models:

- Weekdays model: Neural Network is trained on days that goes from Monday to Friday reacting very well producing a small Losses.

- Week-End model: trained on Saturdays and Sundays, presents a large validation loss. Probably it is due to the fact that the training dataset is very small.

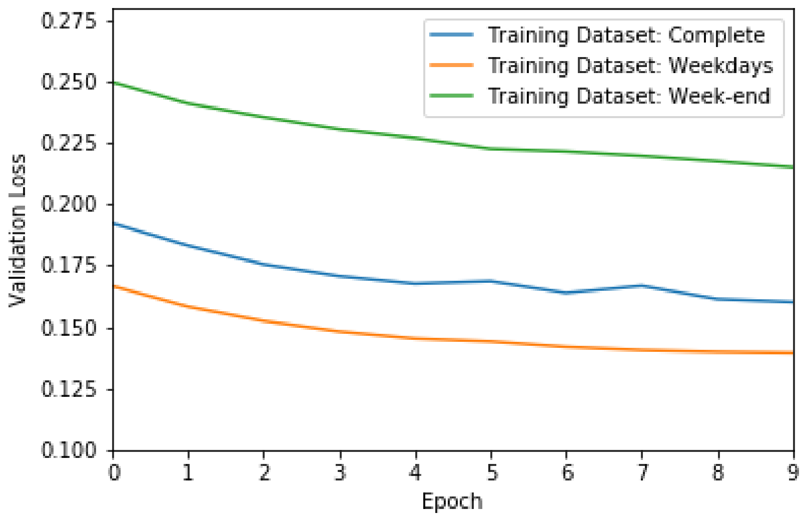


Figure 4.14: No hidden layer analysis validation loss: weekdays and week-end models.

Training dataset	Validation loss	Test loss
Week	0.1601	0.1610
Weekdays	0.1394	0.1480
Week-End	0.2150	0.2228

Table 4.10: No hidden layer analysis validation loss: weekdays and week-end models.

Last result suggests in some way to separate different days in order to take into account the diversity of traffic. For this reason, the entire traffic dataset is modified adding an extra information that identifies the day, as shown in Figure 3.5. In this way, it is obtained the best result up to now.

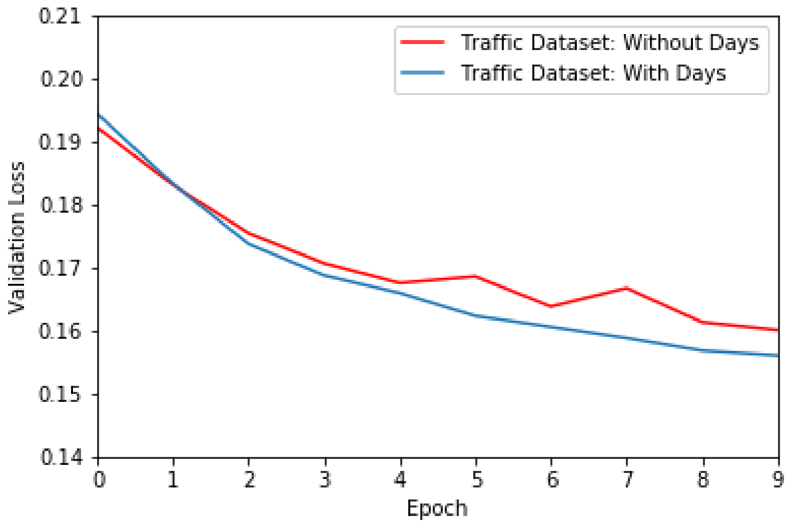


Figure 4.15: No hidden layer analysis validation loss: final dataset.

Day definition	Validation loss	Test loss	Time
No	0.1601	0.1610	3h 22m 11s
Yes	0.1561	0.1563	3h 31m 25s

Table 4.11: No hidden layer analysis evaluations: final dataset

New traffic dataset application produces a gain in terms of losses without implying a significant increment of computational operations. Moreover, up to the 10^{th} epoch there is no oscillation in the validation loss trend, guaranteeing more stability.

4.3.4 L2 regularization analysis

Running training session over 50 epochs for the best found model it is possible to notice, from Figure 4.16, that for large epochs train loss continues to decrease while validation loss starts to oscillate[16]: overfitting problem

[16]In general, simple Neural Network presents achievable performance level limits.

occurs. L2 regularization is exploited in order to provide a more robust model that does not suffer overfitting problem.

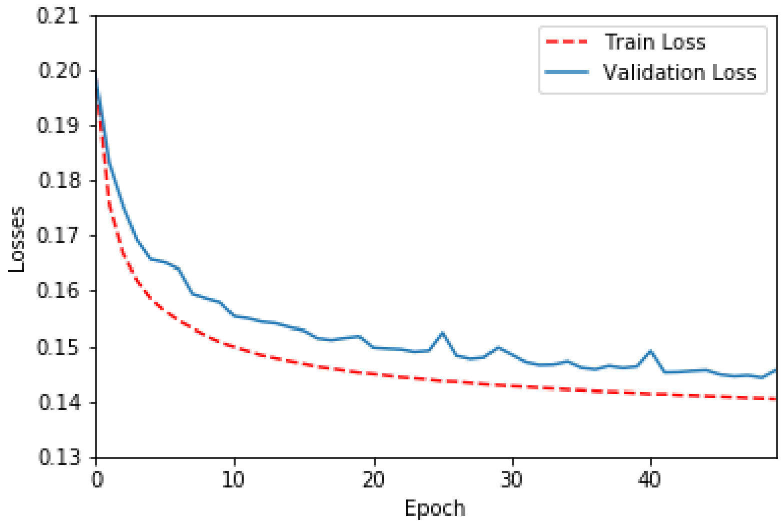


Figure 4.16: No Hidden Layer Analysis Validation Loss: Overfitting.

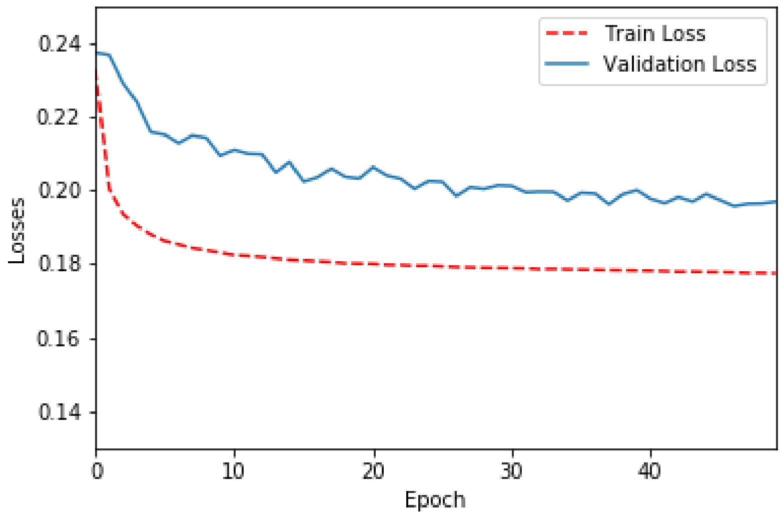


Figure 4.17: No hidden layer analysis validation loss: underfitting.

L2 parameter, from Section 2.1.2, has effect on the flexibility of the obtained model: if it is equal to 0 the penalty term has no effect, obtaining only the minimization of the cost function; if it is too large the penalty term grows very much producing an underfitting problem. As matter of fact, setting the regularization parameter equal to 0.1 it is possible to check from Figure 4.17 the described underfitting problem.

Changing the L2 parameter, in order to get a more reliable model, the goal is to find the right one in such a way that validation loss oscillations do not occur with high frequency.

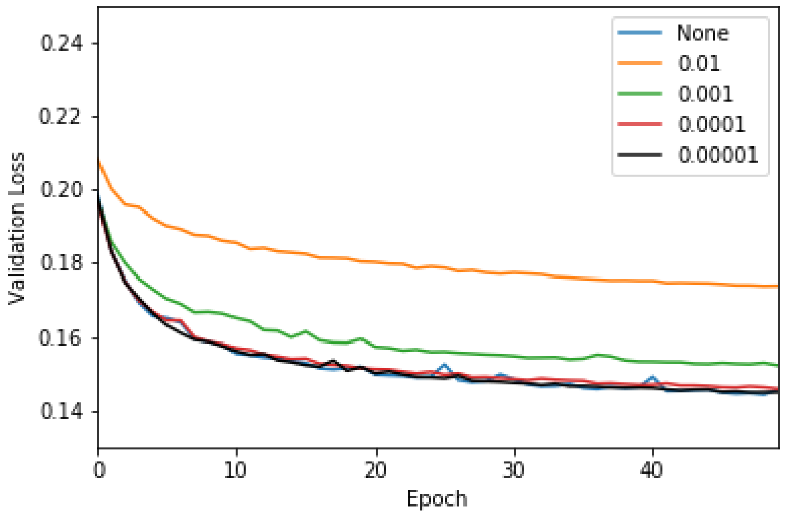


Figure 4.18: No hidden layer analysis validation loss:
L2 regularization.

Plot shows the different behaviors: smaller L2 parameters allow to get values very close to the ones produced by the model with no regularization providing a more stable trend while the larger ones are very far from the optimal performance.

The following table shows numerical results produced applying L2 regularization.

L2 parameter	Validation loss	Test loss	Time
None	0.1456	0.1458	19h 17m 23s
0.1	0.0.1969	0.1957	19h 20m 41s
0.01	0.1738	0.1722	18h 58m 55s
0.001	0.1522	0.1523	19h 2m 1s
0.0001	0.1459	0.1461	18h 59m 24s
0.00001	0.1449	0.1452	19h 11m 53s

Table 4.12: No hidden layer analysis evaluations:
L2 regularization.

4.4 One hidden layer analysis

To get higher performance models it is introduced a hidden layer between input and output layers. This leads to another level of complexity since the number of learning parameters increases very much, quadratic in the worst case.

Datasets used in this section are:

- Training dataset: traffic data of two weeks, from November 4^{th} to November 24^{th}, 2013.
- Validation dataset: from Training dataset, traffic data of Saturday, November 9^{th} and Wednesday, November 13^{th}, 2013.
- Test dataset: traffic data of the week from November 25^{th} to December 1^{st}, 2013.

Hyper-parameters are set as follows:

- Batch size: 1024.
- Epochs number: 10.
- Learning rate: 0.001.
- Neurons:
  - Input Layer: [64, 128, 256, 512, 1024].

– Hidden Layer : [64, 128, 256, 512, 1024][17].

– Output Layer: 1.

Both batch size and epochs hyper-parameters are imposed by computational limits. For each layer it is used a ReLU activation function, except the output one that exploits linear activation function in order to provide a continuous value that can range all over the real domain.

4.4.1 Neurons analysis

At first, exploiting results obtained in "No hidden layer analysis", this study focuses on the research of best Neural Network configuration in terms of neurons number.

Models evaluation starts from the Neural Network in which the input layer is made up of 1024 neurons. A preliminary operation, called batch normalization, is done in order to obtain a more reliable model, improving stability and learning speed. This technique consists in the normalization of a layer output data (in this case, the input layer), before to go in input to the next layer (in this case, hidden layer). In this way, the input data distribution, at each layer, is much more reasonable with respect to the original one that may present coefficients concentration. As proof, considering the Neural Network composed by 1024 neurons for both input layer and hidden layer, the following result on the validation loss is obtained:

[17]Considered number of hidden neurons is always smaller or equal to input neurons

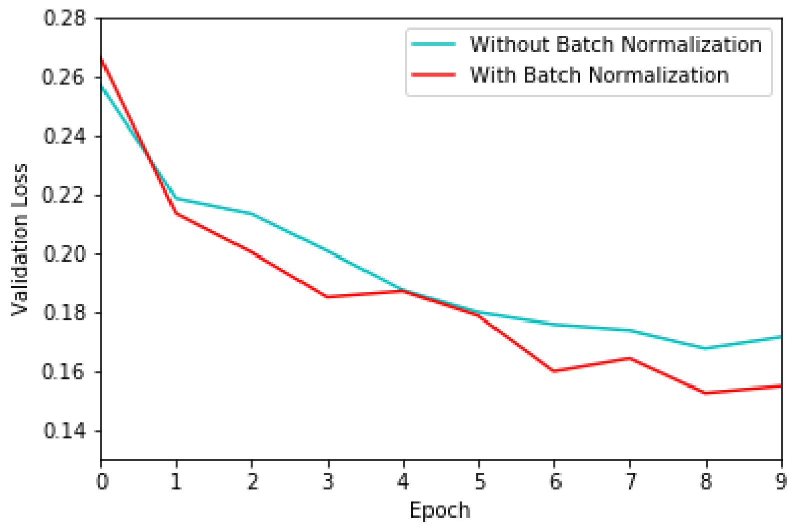


Figure 4.19: One hidden layer analysis validation loss:
batch normalization

The size of the Neural Network, which depends on the number of hidden neurons, is mostly large since the learning parameters can go from 80,001 up to 1,064,961.

Hidden neurons	Learning parameters
64	80,001
128	145,665
256	276,993
512	539,649
1024	1,064,961

Table 4.13: One hidden layer analysis learning parameters:
input Layer = 1024 Neurons.

Checking the validation loss plot stands out how the larger is the network the more unstable is the behavior of the model. In particular, for 1024 and 512 hidden neurons cases, the oscillations become significant. This is due to to the fact that these networks are very large and overfitting phenomena are more probable to occur.

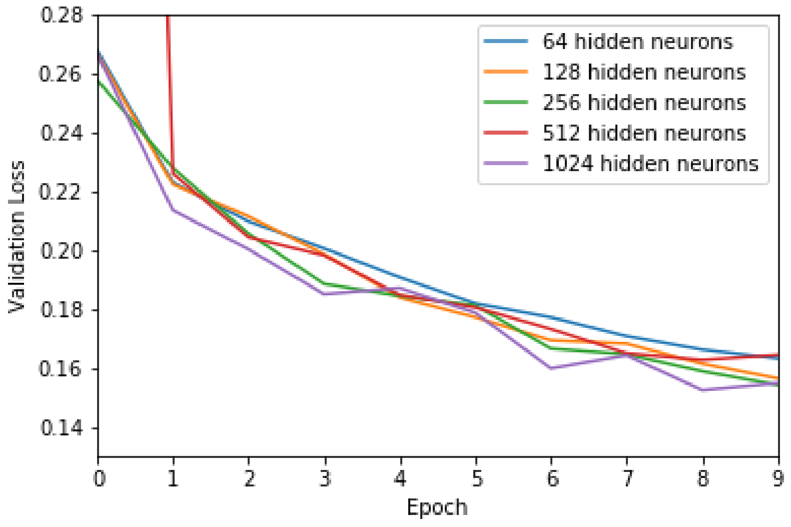


Figure 4.20: One hidden layer analysis validation loss:
input layer = 1024 neurons.

In terms of final losses, values seems to be not so better compared to the ones achieved in "No hidden layer analysis". What seems to be different is the speed through which the model improves, epoch after epoch. Indeed, extending the learning process over more epochs an improvement could be obtained. Last consideration, not least, is the computational time: it becomes larger and always more significant as the network size increases.

Hidden neurons	Validation loss	Test loss	Time
64	0.1631	0.1659	5h 31m 10s
128	0.1563	0.1570	6h 25m 56s
256	0.1541	0.1559	7h 35m 40s
512	0.1643	0.1667	14h 9m 42s
1024	0.1547	0.1567	18h 0m 6s

Table 4.14: One hidden layer analysis evaluations:
input layer = 1024 neurons.

512 input layer neurons case is characterized by smaller losses, with computation time clearly reduced. Despite the best loss values are obtained using 256 hidden neurons, the 128 hidden neurons configuration, as the 64, marks out a continuous decreasing trend.

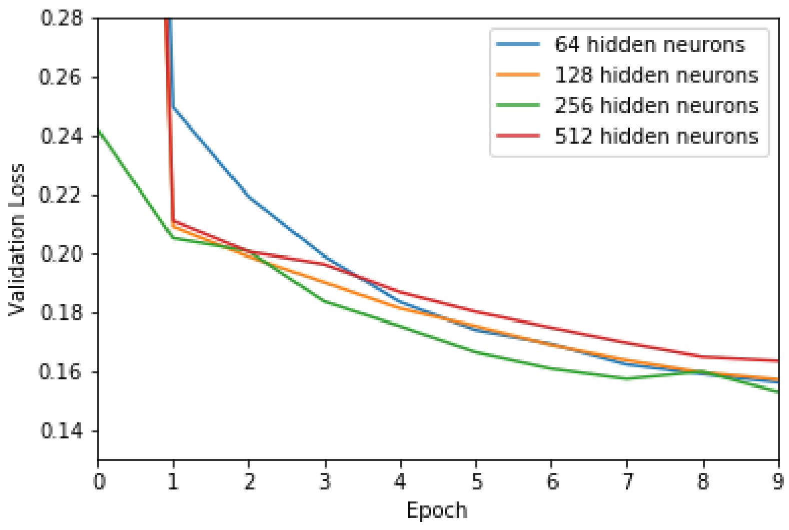


Figure 4.21: One hidden layer analysis validation loss:
input layer = 512 neurons.

Hidden neurons	Validation loss	Test loss	Time
64	0.1560	0.1560	4h 21m 8s
128	0.1570	0.1574	4h 41m 30s
256	0.1528	0.1541	6h 8m 15s
512	0.1633	0.1651	7h 22m 33s

Table 4.15: One hidden layer analysis evaluations:
input layer = 512 neurons.

Hidden neurons	Learning parameters
64	40,065
128	72,961
256	138,753
512	270,377

Table 4.16: One hidden layer analysis learning parameters:
input layer = 512 neurons.

The behavior of each 256 input layer neurons study case is almost the same, and built models return very close loss values. This is probably due to the fact that sizes of the Neural Networks are of the same order of magnitude.

Hidden neurons	Learning parameters
64	20,097
128	36,609
256	69,633

Table 4.17: One hidden layer analysis learning parameters: input layer = 256 neurons.

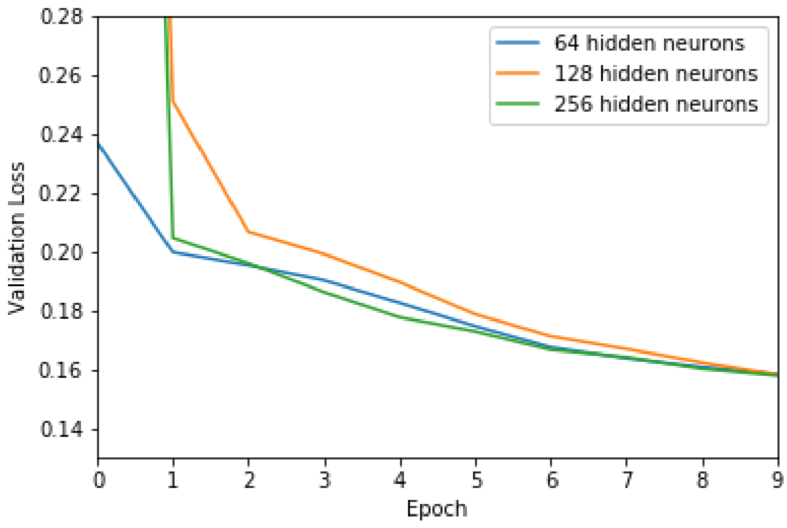


Figure 4.22: One hidden layer analysis validation loss: input layer = 256 neurons.

Hidden neurons	Validation loss	Test loss	Time
64	0.1582	0.1584	3h 26m 28s
128	0.1581	0.1585	3h 34m 24s
256	0.1577	0.1583	4h 9m 59s

Table 4.18: One hidden layer analysis evaluations: input layer = 256 neurons.

In last two cases, Neural Networks of 128 and 64 input layer neurons are quite small.

Hidden neurons	Learning parameters
64	10,113
128	18,433

Table 4.19: One hidden layer analysis learning parameters: input Layer = 128 neurons.

Hidden neurons	Learning parameters
64	5,121

Table 4.20: One hidden layer analysis learning parameters: input Layer = 64 neurons.

As a matter of fact, they behaves approximately as a no hidden layer Neural Network with a validation loss that is going to stabilize. A significant difference is that there are no oscillations: this is due to the fact that small Neural Network are more robust, in general, against the overfitting problem.

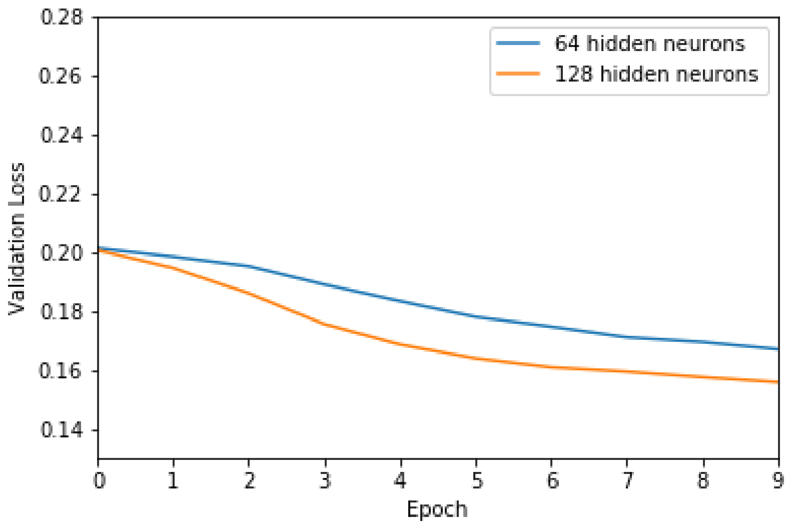


Figure 4.23: One hidden layer analysis validation loss: input layer = 128 neurons.

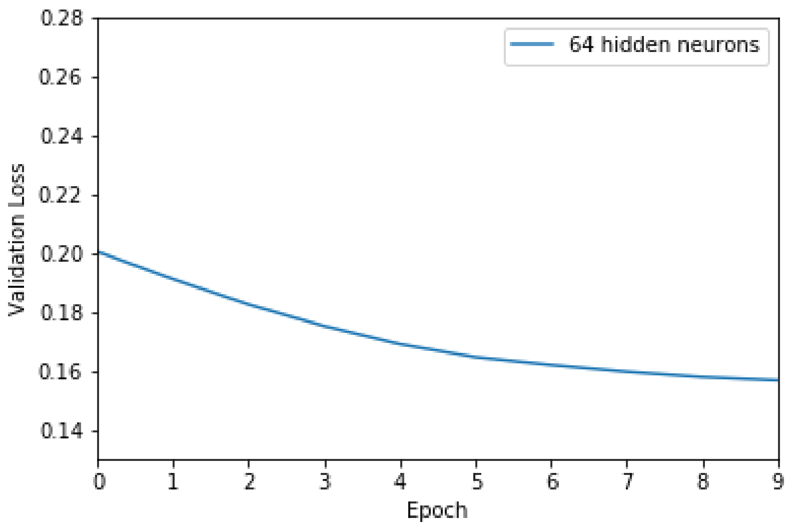


Figure 4.24: One hidden layer analysis validation loss:
input layer = 64 neurons.

Hidden neurons	Validation loss	Test loss	Time
64	0.1669	0.1670	3h 4m 32s
128	0.1557	0.1562	3h 16m 36s

Table 4.21: One hidden layer analysis evaluations:
input layer = 128 neurons.

Hidden neurons	Validation loss	Test loss	Time
64	0.1566	0.1577	2h 56m 26s

Table 4.22: One hidden layer analysis evaluations:
input layer = 64 neurons.

In each case analyzed, results are similar to the one obtained previously with the Neural Network composed just of the input and the output layer. What is different is the speed through which the model learns, improving faster the performance. This effect may be clearer running the training sessions over more epochs.

In the following scatter plot are reported all the test losses obtained.

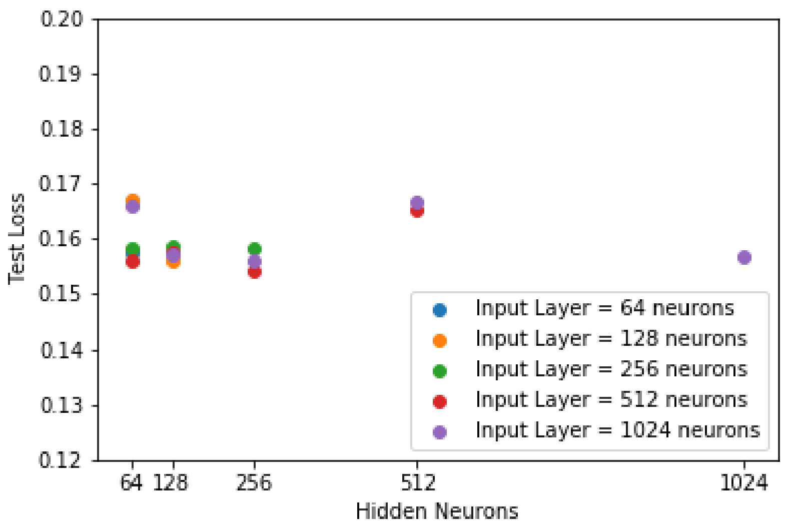


Figure 4.25: One hidden layer analysis test loss.

4.4.2 Dropout and L2 regularization analysis

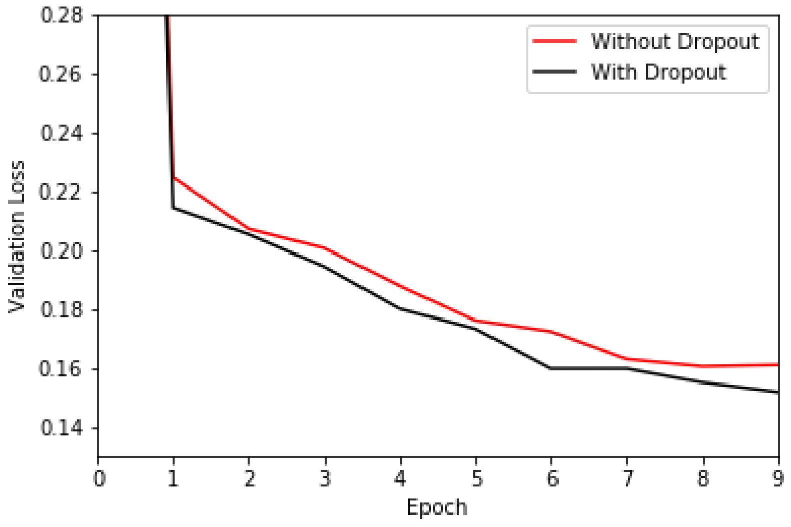
In order to get improvements, the analysis continues focusing on large Neural Networks. In particular, dropout and regularization techniques are applied on Neural Networks configured according to the following table:

Input neurons	Hidden neurons	Learning parameters
512	256	138,753
512	512	270,377
1024	256	276,993
1024	512	539,649
1024	1024	1,064,961

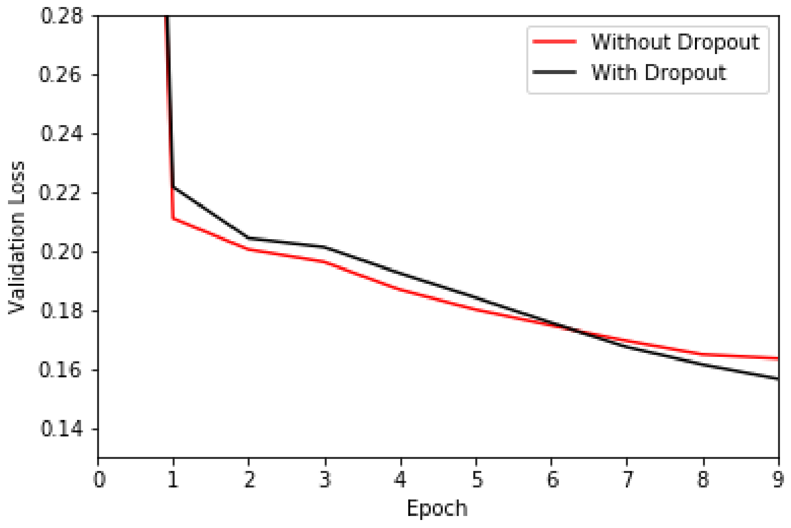
Table 4.23: One hidden layer analysis learning parameters:
input layer = 64 neurons.

Dropout

Dropout is applied to the hidden layer: for each batch learning session some random neurons are zeroed-out according to a fraction defined by the dropout rate. Setting the rate to 0.2, the following results are obtained:

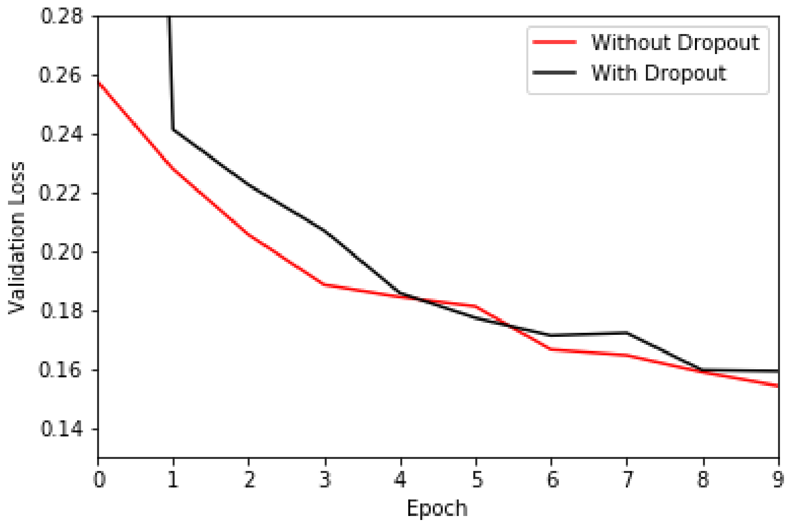


(a) Hidden neurons = 256

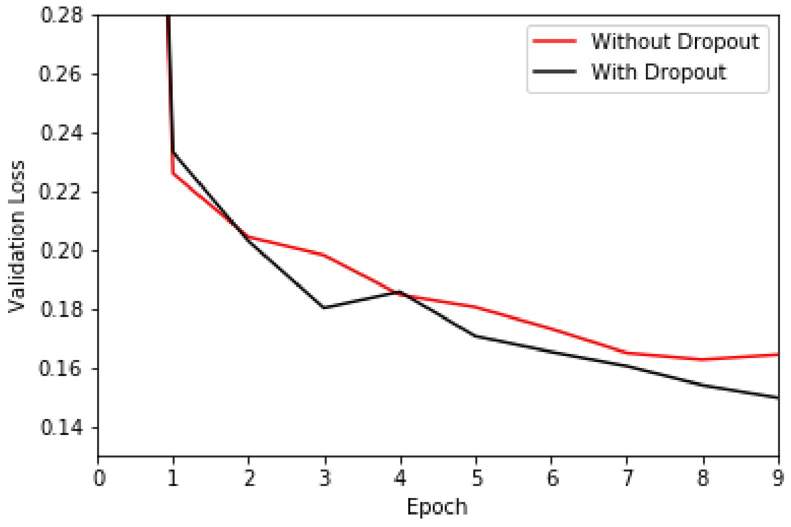


(b) Hidden neurons = 512

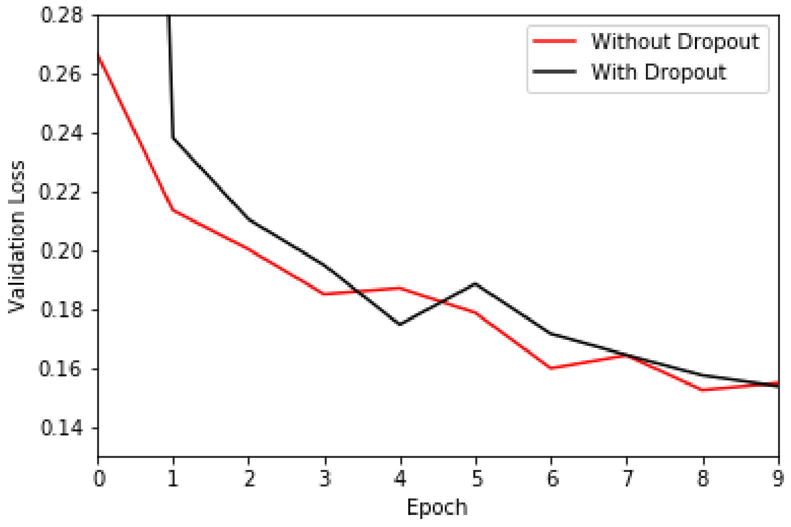
Figure 4.26: One hidden layer analysis validation loss: dropout on 512 input neurons Neural Networks.



(a) Hidden neurons = 256



(b) Hidden neurons = 512



(c) Hidden neurons = 1024

Figure 4.27: One hidden layer analysis validation loss: dropout on 1024 input neurons Neural Networks.

Dropout technique allows to get benefits for Neural Network in which the hidden layer is composed by a conspicuous number of neurons. In fact, in both input layer cases the configuration with 256 hidden neurons does not react very well to the dropout application, both in terms of validation loss trend and performance.

Input neurons	Hidden neurons	Validation loss	Test loss
512	256	0.1555	0.1560
512	512	0.1564	0.1578
1024	256	0.1591	0.1607
1024	512	0.1496	0.1504
1024	1024	0.1536	0.1552

Table 4.24: One hidden layer analysis evaluations: dropout on Neural Networks.

This phenomenon can be better observed grouping all validation losses:

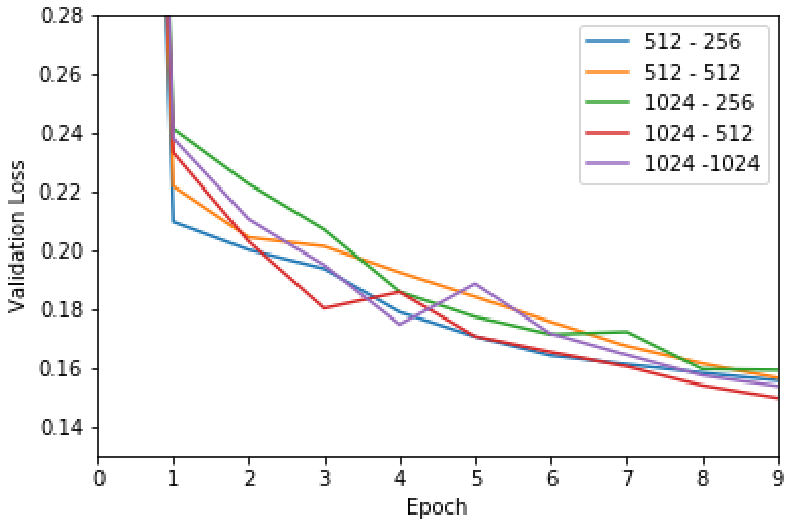
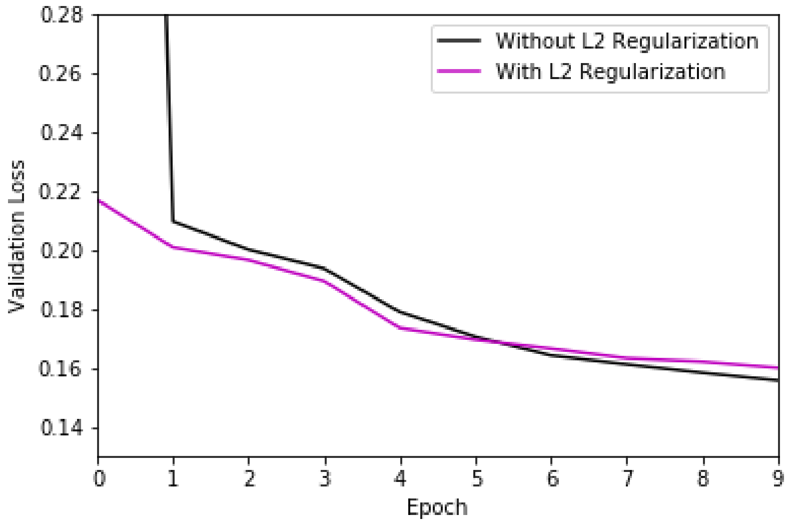


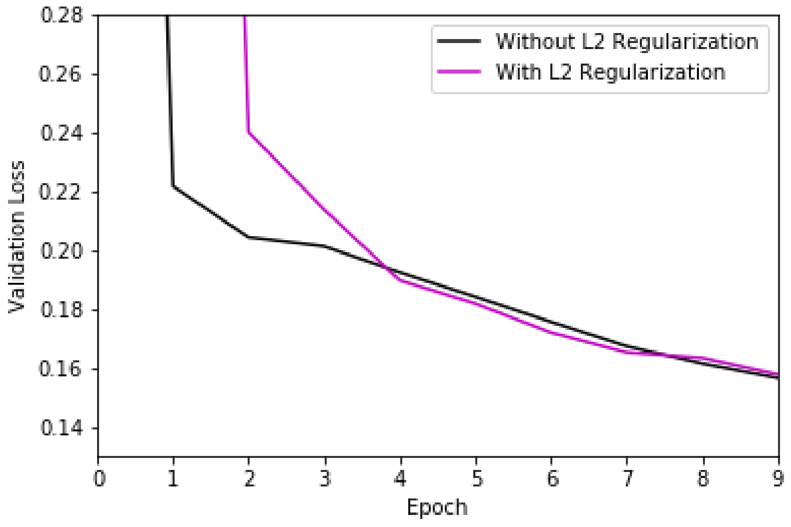
Figure 4.28: One hidden layer analysis validation loss: dropout on Neural Networks.

L2 Regularization

Using these new results, L2 regularization is applied in order to get even more reliable models. Actually, at first glance, the validation loss seems to be affected in a negative manner by the L2 application:

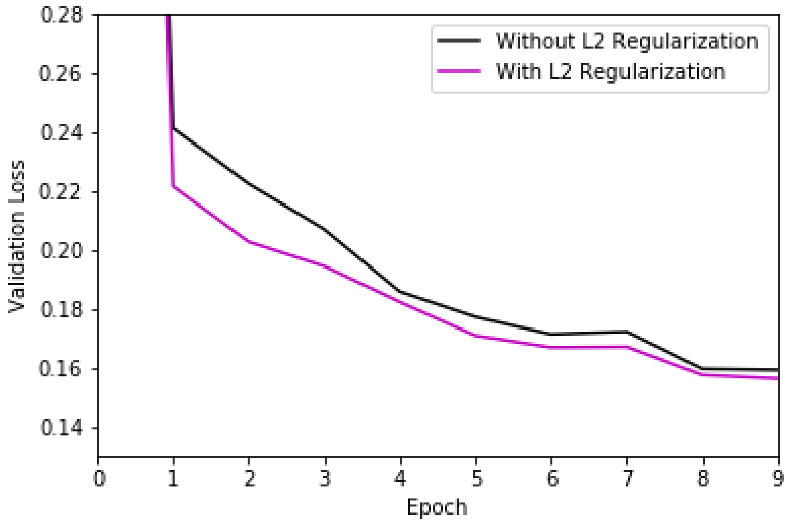


(a) Hidden neurons = 512

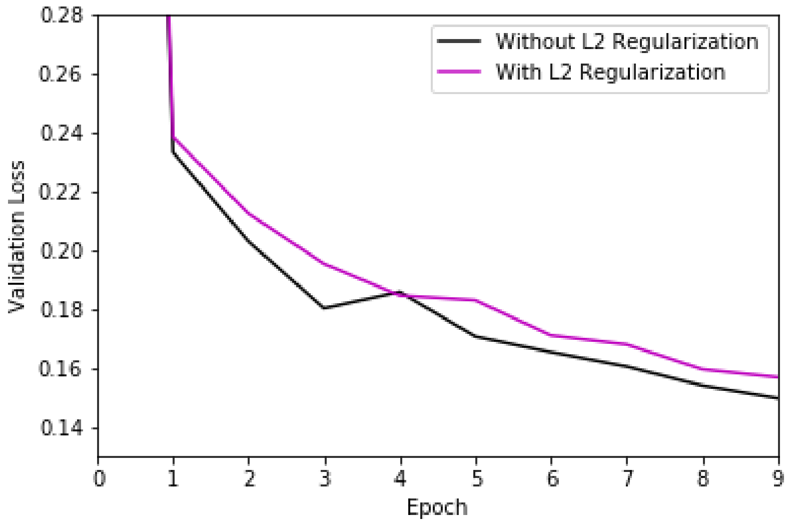


(b) Hidden neurons = 512

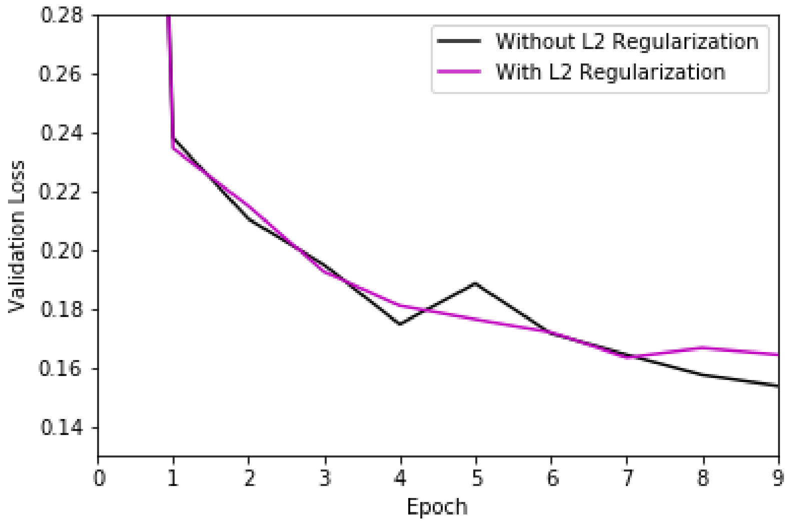
Figure 4.29: One hidden layer analysis validation loss: L2 regularization on 512 input neurons Neural Networks.



(a) Hidden neurons = 256



(b) Hidden neurons = 512



(c) Hidden neurons = 1024

Figure 4.30: One hidden layer analysis validation loss:
L2 regularization on 1024 input neurons Neural Networks.

In order to get an idea of the effect of the regularization on these Neural Networks more epochs are needed. In particular, due the huge computational complexity, the comparison between L2 application and no application performances is done just on the 1024 - 512 Neural Network.

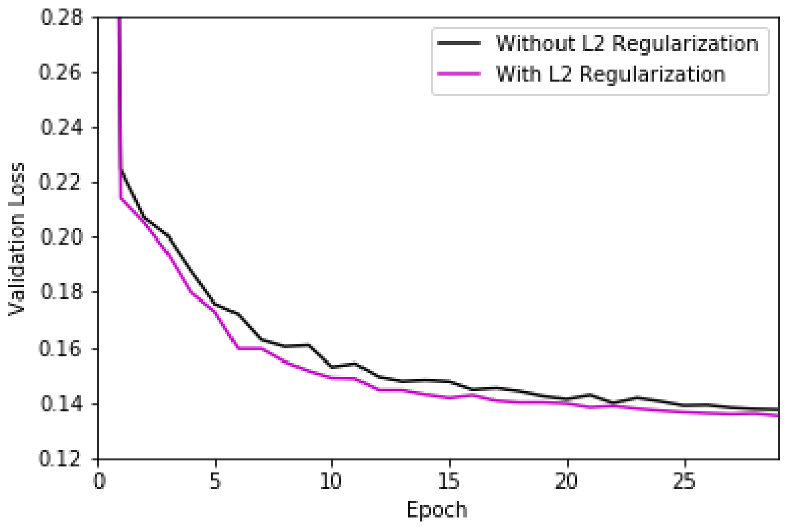


Figure 4.31: One hidden layer analysis validation loss:
L2 regularization effect over 30 epochs.

L2 regularization allows to get small performance improvements and provides more stability and reliability of the model.

Extending learning process of the so defined large Neural Network over 30 epochs, the following results are obtained:

Input neurons	Hidden neurons	Validation Loss	Test Loss
512	256	0.1429	0.1439
512	512	0.1413	0.1420
1024	256	0.1382	0.1385
1024	512	0.1353	0.1356
1024	1024	0.1306	0.1314

Table 4.25: One hidden layer analysis evaluations:
L2 regularization - I.

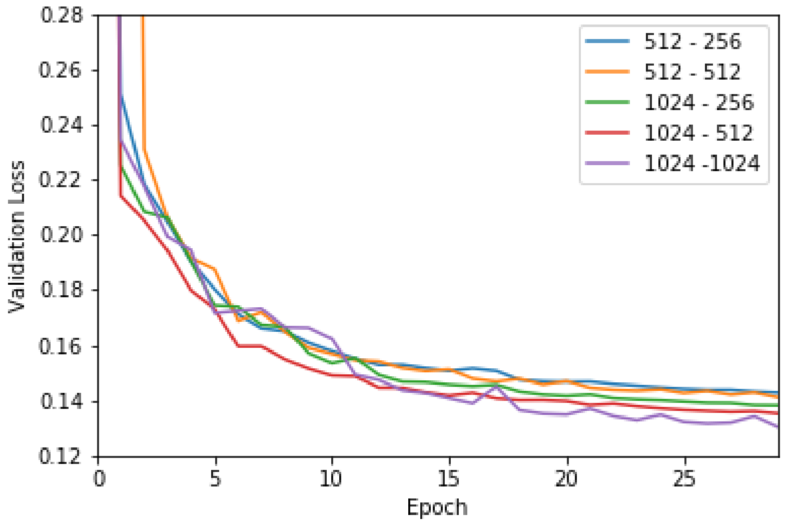


Figure 4.32: One hidden layer analysis validation loss:
L2 regularization.

Therefore, the larger is the Network the better will be the loss result. By contrast, enlarging the number of epochs produces a significant increase of computations, requiring too much time.

Input neurons	Hidden neurons	Time
512	256	22h 15m 31s
512	512	1d 4h 5m 57s
1024	256	1d 9h 14m 21s
1024	512	1d 23h 41m 30s
1024	1024	2d 17h 21m 39s

Table 4.26: One hidden layer analysis evaluations:
L2 regularization - II.

This means that Deep Neural Networks analysis produce very likely a better performance in terms of losses while becomes very expensive in terms of computational time. For this reason, are needed more powerful computational resources.

Chapter 5

MEC Dimensioning

5.1 Multi-access Edge Computing for 5G

Fifth Generation Mobile Network (5G) is going to bring with it some important improvements with respect to older generations. Its main use-cases are:

- Enhanced Mobile Broadband: mobile users will be served with very high bit rate (1 Gbps for user experience, 10 Gbps fot peak rate)
- Massive Machine-Type Communication: it will provide communication to a huge number of connected devices.
- Ultra-Reliable Low Latency Communication: connections will be characterized by ultra low message error rate (FER of the order 10^{-9}) and extremely low end-to-end latency (smaller than 1 ms).

From the network architectural point of view are introduced some concepts and technologies: this is the case of Multi-access Edge Computing. The idea behind MEC is to support could computing capabilities at the edge of the mobile network[18] in such a way that data process is done in proximity of the users. In this way, latency becomes significantly smaller and higher speed are achieved. This is due mainly to congestion reduction in core and backhaul networks, produced by traffic that is exponentially growing through different kind of application as IoT, UHD and gaming.

[18]In general, MEC cloud computing capabilities are implemented through data center on edges nodes as antenna infrastructures, called gNodeB (gNB) in 5G.

Moreover, it allows to enable IT service environment in order to get local context information as RAN (Radio Access Network) analysis or traffic characteristics in order to improve efficiency.

5.2 Deep Learning for MEC Dimensioning

MEC dimensioning is a well posed problem: to get higher performance in terms of rate and latency in 5G, cloud computing resources have to be placed as closer as possible to the user. It is easy to understand that over-provisioning mobile network with MEC platforms at each gNB can be very expensive. By means of the study of the Internet traffic activity and the creation of a prediction model faced out in the previous Chapter, the idea becomes to estimate the traffic request[19] to check where it is higher in order to allocate MEC resources according to an optimization algorithm.

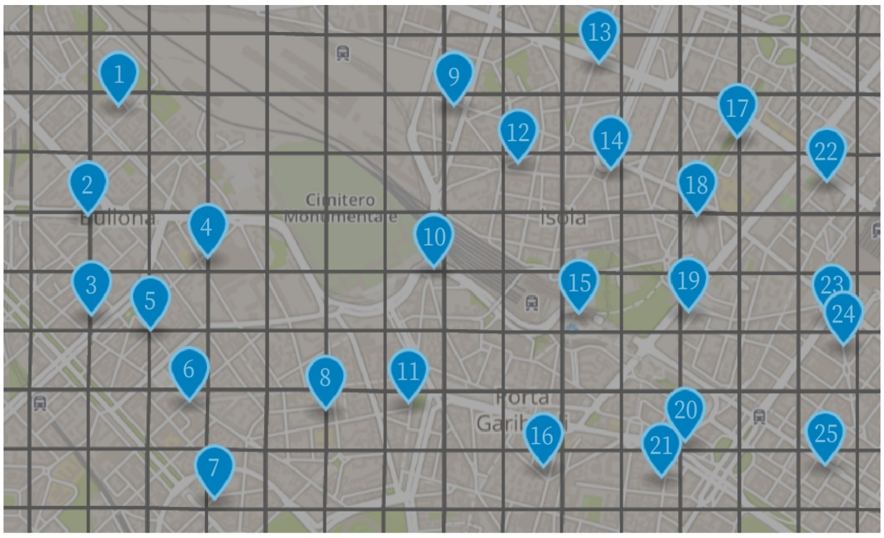


Figure 5.1: Milano Porta Garibaldi station area.

For this purpose it is considered an area of Milano, around Porta Garibaldi station, through the grid dataset (Section 3.1.1). Then the location of gNBs

[19]Nowadays, Internet traffic network defines the greatest part of total traffic activities.

infrastructure, taken from [12][20] is added. In order to get traffic distribution, it is defined also a possible cell coverage area for each tower, respecting squares limits.

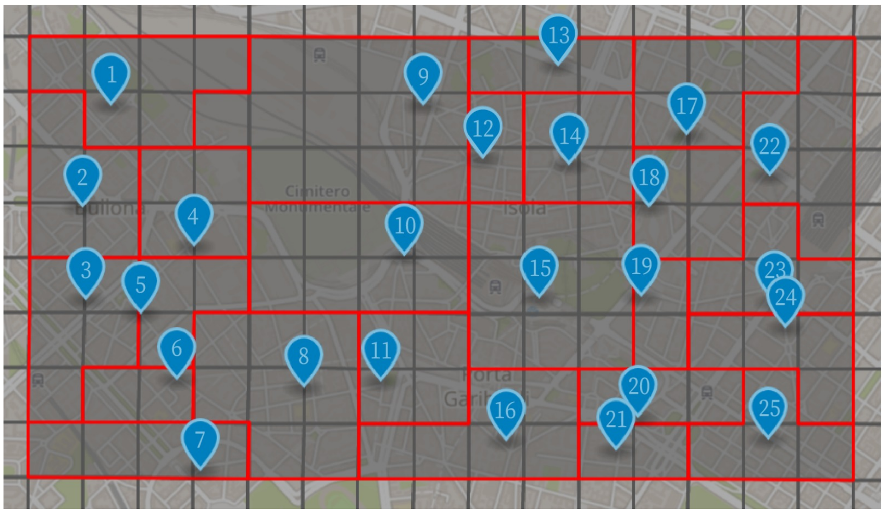


Figure 5.2: Milano Porta Garibaldi station area divided in cells.

gNBs are connected to a datacenter forming a star topology. This element is in charge of communications.

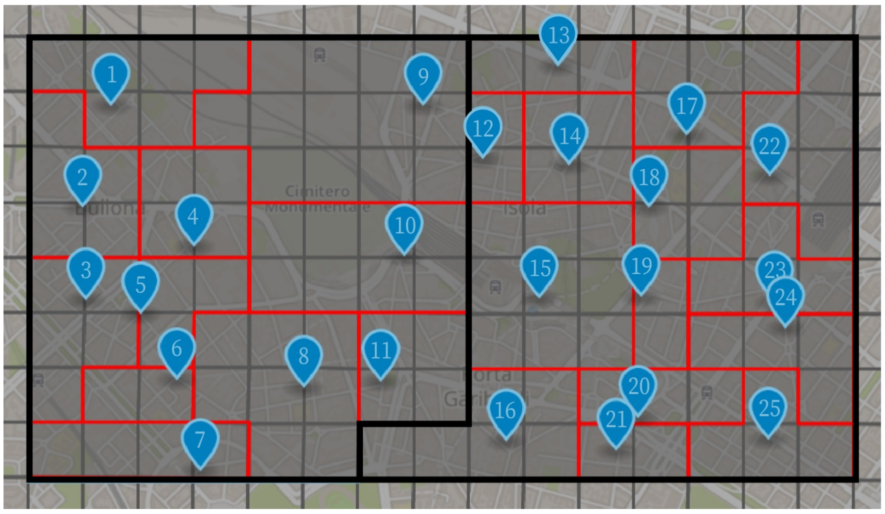


Figure 5.3: Milano Porta Garibaldi station area divided in topologies.

In this specific case, two subsets are defined over the area: the first one

[20][12] provides the current infrastructures location that will be used also by 5G.

takes infrastructures belonging to the left part of the map while the second is composed by the rest.

5.2.1 Optimization model

Optimization model identifies number and location of gNB towers that can host an installation of a MEC platform in function of the traffic demand coming from users.

Assuming that an area is completely served and covered by cells that are fed by A tower infrastructures. This area can be equipped with up to M MEC datacenters, each co-located with a gNB infrastructure site, at the cost of installation/maintenance of m_i. Each MEC platform has capacity K_i. Load in units of traffic resulting from service requests coming from all cells served by antenna j is denoted as l_j, with $j = 0, \ldots, A$. If a cell is not served by a gNB tower with a co-located MEC, (i.e., antenna j needs to access the MEC at antenna i), its traffic will incur a penalty cost c_{ij}, which is meant to represent a combination of effects stemming from the additional latency and possible bottlenecks that it may encounter as it connects to a MEC co-located at a nearby antenna. If the traffic comes from a cell whose infrastructure is co-located with a MEC, its penalty cost would be expressed as $c_{ij} = 0$. It is to be remarked that in this model does not taking into account the traffic that is exchanged with the core network but only the traffic that remains local and needs to be routed through a MEC (either co-located with the gNB or on a nearby one).

Three additional variables are defined:

- x_i: binary decision variable denoting the presence of a MEC datacenter at a gNB site (equal to 0 if no MEC is colocated; equal to 1 if MEC is co-located).
- y_{ij}: binary auxiliary variable denoting whether MEC i is connected to tower j (equal to 0 if it is not connected; equal to 1 if it is connected).
- f_{ij}: real decision variable denoting the traffic flow (in units of traffic)

from tower j to MEC i.

So, the goal of the optimization is to find the minimum cost of installation that at the same time minimizes the penalties incurred in by having multiple cells using distant MECs for their local traffic:

$$\min_{x,f} \sum_{i=1}^{M} \left(x_i m_i + \sum_{j=1}^{A} f_{ij} c_{ij} \right)$$

It needs to be solved taking into account several constraints:

- Variable constraints:
  - the auxiliary variable y_{ij} is 0 when the corresponding MEC location is empty
    $$y_{ij} \leq x_i \quad \forall i, j$$
  - traffic flow can only appear if the MEC i is connected to tower j, and then it cannot be larger than the load from service requests from that tower
    $$f_{ij} \leq y_{ij} l_i \quad \forall i, j$$
- Capacity constraints:
  - the sum of all traffic flows entering a MEC datacenter cannot exceed the capacity of the MEC
    $$\sum_{i=1}^{M} f_{ij} \leq K_i \quad \forall i$$
- Flow constraints:
  - traffic flows cannot be negative
    $$f_{ij} \geq 0 \quad \forall i, j$$
- Flow conservation:

- all traffic flows resulting from service requests at tower j must reach a MEC

$$\sum_{i=1}^{M} f_{ij} = l_j \quad \forall j$$

- all traffic from all service requests in the network must account for all the traffic flows found in the network

$$\sum_{j=1}^{A} l_i = \sum_{i=1}^{M} \sum_{j=1}^{A} f_{ij} \quad \forall i, j$$

This is a Mixed-Integer Linear Programming category (MILP). These problems are generally NP-hard, so a solution can be found only for relatively small values of A and M.

5.2.2 Dimensioning

Described optimization model is applied with the following optimization parameters:

- Installation cost m_i: [5, 10].
- Penalty cost c_{ij}:
  - 0 if MEC datacenter is within the gNB infrastructure from which the traffic comes.
  - 1 if MEC is co-located on a different gNB infrastructure that is possible to reach just with one hop (belong to the same star topology).
  - 2 if MEC is co-located on a different gNB infrastructure that is possible to reach just with mote than one hop (belong to a different star topology).
- MEC capacity K_i: [3, 6, 9] Gb/s.

The efficiency of the prediction model in MEC dimensioning context is evaluated applying optimization on two scenarios:

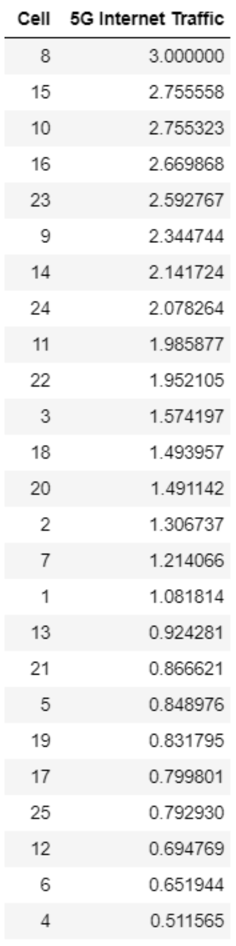
- Worst case: from traffic dataset are extracted Internet traffic peaks of each cell over the entire period that goes from November 25^{th} to December 31^{st}, 2013.
- Predicted case: it is used a prediction model in order to get an estimation of the Internet traffic over a complete week, November 25^{th} to December 1^{st}, 2013. In particular, the model is obtained through a Neural Network with one hidden layer and with hyper-parameters set as follows:
  - Batch size: 1024.
  - Epochs number: 30.
  - Learning rate: 0.001.
  - Neurons:
    * Input layer: 1024.
    * Hidden layer : 1024.
    * Output layer: 1.

  Finally, is taken an average over the traffic of each cell for a more reliable dimensioning.

Internet traffic analyzed so forth describes the activity as slot of bytes. In order to express it directly in terms of bytes it is exploited the fact that each connection between towers and the datacenter of the star topology has capacity 3 Gb/s, value used as normalization factor for traffic over all the cells. The hypothesis done in order to get this normalization factor is that each gNB tower provides communication between users and the rest of the network by means of three cells. Each one of these cells has 1 Gb/s capacity[21].

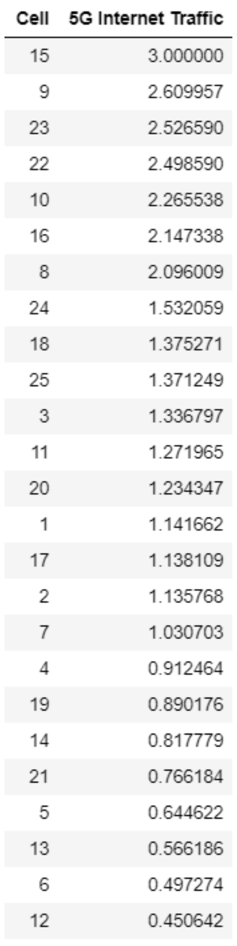
[21]The communication between each cell and the gNB tower a Gb-Ethernet cable.

So, the l_j optimization parameter is defined, for each cell, according to the following values:



Cell	5G Internet Traffic
8	3.000000
15	2.755558
10	2.755323
16	2.669868
23	2.592767
9	2.344744
14	2.141724
24	2.078264
11	1.985877
22	1.952105
3	1.574197
18	1.493957
20	1.491142
2	1.306737
7	1.214066
1	1.081814
13	0.924281
21	0.866621
5	0.848976
19	0.831795
17	0.799801
25	0.792930
12	0.694769
6	0.651944
4	0.511565

(a) Worst case traffic



Cell	5G Internet Traffic
15	3.000000
9	2.609957
23	2.526590
22	2.498590
10	2.265538
16	2.147338
8	2.096009
24	1.532059
18	1.375271
25	1.371249
3	1.336797
11	1.271965
20	1.234347
1	1.141662
17	1.138109
2	1.135768
7	1.030703
4	0.912464
19	0.890176
14	0.817779
21	0.766184
5	0.644622
13	0.566186
6	0.497274
12	0.450642

(b) Predicted case traffic

Figure 5.4: 5G normalized traffic.

5.2.3 Results

Optimization results indicate, for each gNB infrastructure, if MEC datacenter has to be co-located on that node or not. Although installation cost m_i is the most critical parameter, it is reported a unique case for both values since results are identical.

gNB	MEC co-location
1	No
2	Yes
3	Yes
4	No
5	No
6	No
7	No
8	Yes
9	Yes
10	Yes
11	Yes
12	Yes
13	No
14	Yes
15	Yes
16	Yes
17	No
18	Yes
19	No
20	No
21	No
22	Yes
23	Yes
24	Yes
25	No
Total MEC: 14	

(a) Worst case.

gNB	MEC co-location
1	No
2	No
3	Yes
4	No
5	No
6	No
7	No
8	Yes
9	Yes
10	Yes
11	Yes
12	Yes
13	No
14	No
15	Yes
16	Yes
17	No
18	Yes
19	No
20	Yes
21	No
22	Yes
23	Yes
24	No
25	No
Total MEC: 12	

(b) Predicted case.

Table 5.1: MEC co-location site: MEC capacity = 3 Gb/s.

gNB	MEC co-location
1	No
2	No
3	No
4	No
5	No
6	No
7	No
8	Yes
9	Yes
10	Yes
11	No
12	Yes
13	No
14	No
15	Yes
16	Yes
17	No
18	No
19	No
20	No
21	No
22	No
23	Yes
24	No
25	No
Total MEC: 7	

(a) Worst case.

gNB	MEC co-location
1	No
2	No
3	No
4	No
5	No
6	No
7	No
8	Yes
9	Yes
10	Yes
11	No
12	No
13	No
14	No
15	Yes
16	Yes
17	No
18	No
19	No
20	No
21	No
22	Yes
23	No
24	No
25	No
Total MEC: 6	

(b) Predicted case.

Table 5.2: MEC co-location site: MEC capacity = 6 Gb/s.

gNB	MEC co-location
1	No
2	No
3	No
4	No
5	No
6	No
7	No
8	Yes
9	No
10	Yes
11	No
12	Yes
13	No
14	No
15	Yes
16	Yes
17	No
18	No
19	No
20	No
21	No
22	No
23	No
24	No
25	No
Total MEC: 5	

(a) Worst case.

gNB	MEC co-location
1	No
2	No
3	No
4	No
5	No
6	No
7	No
8	No
9	Yes
10	Yes
11	No
12	No
13	No
14	No
15	Yes
16	Yes
17	No
18	No
19	No
20	No
21	No
22	No
23	No
24	No
25	No
Total MEC: 4	

(b) Predicted case.

Table 5.3: MEC co-location site: MEC capacity = 9 Gb/s.

MEC capacity represents a very important parameter: the higher it is the smaller is the number of installed MECs platforms. So, there is an important trade-off between cost installation and capacity.

Most important consideration is that comparing the two cases it is possible to notice how worst case optimization places a larger number of MEC datacenter. Therefore, using prediction and applying optimization algorithm on corresponding estimated traffic data it is possible to dimension MECs over a specific area satisfying efficiently user requests.

Conclusions

This thesis work particularly focused on the research of a prediction model able to estimate Internet traffic activity over a geographical area, which is Milan. This study is then addressed to the MEC dimensioning problem. In particular, given an area, it is proved that predicted traffic allows to get a number reduction of MEC infrastructures installation with respect to a dimensioning done on worst case hypothesis.

The prediction model is obtained through Neural Networks implementation. Specifically, a first analysis is done on a Neural Network with just input and output layer in order to look for the best hyper-parameters that allow minimization of the error produced on each predicted element. Results revealed that smaller batch size allow to get benefits in terms of losses, despite larger computation are needed. Moreover, larger networks (higher number of neurons) perform better, through right datasets definition and regularization techniques application managing. Finally, by means of "One hidden layer analysis" the best prediction model is found: even if results are not optimal, they showed that the more complex the Neural Network is the better are the performances achieved in terms of losses.

This suggests to go toward Deep Neural Networks implementation to get better loss results. The problem is that they will require huge computation resources. As a matter of fact, best found model (composed of just input, hidden and output layer) required up to a few days of computation.

Therefore, the study could be further examined moving toward Deep Neural Networks with the introduction of several hidden layers that could improve significantly model performance in terms of loss although requiring much higher computational resources.

Bibliography

[1] https://www.webeducenter.it/corso-machine-learning/

[2] https://www.researchgate.net/figure/Linear-Regression-model-sample-illustration_fig3_333457161

[3] https://databricks.com/glossary/neural-network

[4] http://neuralnetworksanddeeplearning.com/

[5] https://pathmind.com/wiki/neural-network

[6] https://www.researchgate.net/figure/Deep-Learning-neural-network_fig2_311966233

[7] https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss

[8] https://blog.paperspace.com/content/images/size/w2000/2018/05/convex_cost_function.jpg

[9] https://d2l.ai/chapter_multilayer-perceptrons/underfit-overfit.html

[10] http://neuralnetworksanddeeplearning.com/chap3.html

[11] https://dandelion.eu

[12] https://www.inwit.it/it/copertura