POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

DEEP NEURAL NETWORKS FOR TIME SERIES FORECASTING BEYOND TRANSFORMERS

Tesi di Laurea Magistrale in
Computer Science Engineering - Ingegneria Informatica

Riccardo Ughi, 10590759

Abstract: Human culture has always wondered how to know the future in advance. In recent years, a specific subject of Machine Learning studies is precisely this, trying to grasp the future from the study of time series with the use of neural networks. In other words, it aims to find neural network models applicable to time series that are able to calculate the trend of an unknown phenomenon from other known parts of it. Time series are sequential data in which an input data vector represents time. It is clear that if it were possible to calculate time series values before knowing them through real-world observation, it would mean being able to predict the future, and this would certainly have great appeal and obvious practical interest. Time series forecasting sets itself precisely this goal, that of calculating the future.
There are numerous active lines of research dealing with this using different techniques. In particular, numerous studies develop models derived from the Transformer, which was originally invented for linguistic analysis. Many researchers have therefore found it natural to try to extend its use to other fields and in particular to time series in order to predict future trends.
Several researches foresee a growing complication of the models, in a sort of emulation of what happened in other areas, such as the classification of images and Neural Language Processing (NLP) in which a growing complexity of Deep Learning models can correspond to a better Time series forecasting sets itself precisely this goal, that of calculating the future!.
We have also developed models capable of making predictions for extremely long periods, up to 5376 values in some cases. We have shown that even for such long periods, the results are good if the data set has a constant and regular periodicity. From an interpretative point of view, this means that if there is regularity in the data, the Neural Network models are able to adapt, being able to obtain the necessary information when the values are periodic. The models we proposed, realised with substantially reduced levels of complexity compared to other studies, achieve comparable and in many cases superior performance compared to the state of the art. We have also shown that if the regularity condition does not exist, long-term predictions are more problematic and more often than not impossible.

Advisor:
Prof. Matteo Matteucci

Co-advisors:
Eng. Eugenio Lomurno

Academic year:
2021-2022

Key-words: Machine Learning, Deep learning, Time Series, Transformers, Neural Network, Forecasting

Abstract in lingua italiana

L'uomo si è sempre domandato come conoscere il futuro in anticipo. Negli ultimi anni, un argomento specifico degli studi di Machine Learning è proprio questo, cercare di cogliere il futuro dallo studio delle serie temporali con l'uso delle reti neurali. In altre parole, si tratta di trovare modelli di reti neurali applicabili alle serie temporali che siano in grado di calcolare la tendenza di un fenomeno sconosciuto a partire da altre parti note di esso. Le serie temporali sono dati sequenziali in cui un vettore di dati in ingresso rappresenta il tempo. È chiaro che se fosse possibile calcolare i valori delle serie temporali prima di conoscerli attraverso l'osservazione del mondo reale, significherebbe essere in grado di prevedere il futuro, e questo avrebbe certamente un grande fascino e un evidente interesse pratico. La previsione delle serie temporali si pone proprio questo obiettivo, quello di calcolare il futuro. Esistono numerose linee di ricerca attive che si occupano di questo aspetto utilizzando tecniche diverse. In particolare, numerosi studi sviluppano modelli derivati dal trasformatore, originariamente inventato per l'analisi linguistica. Molti ricercatori hanno quindi trovato naturale cercare di estendere il suo utilizzo ad altri campi e in particolare alle serie temporali per prevedere le tendenze future.
Diverse ricerche prevedono una crescente complicazione dei modelli, in una sorta di emulazione di quanto avvenuto in altri ambiti, come la classificazione delle immagini e il Neural Language Processing (NLP) in cui a una crescente complessità dei modelli di Deep Learning può corrispondere una migliore previsione delle serie temporali che si pone proprio questo obiettivo, quello di calcolare il futuro!
Abbiamo anche sviluppato modelli in grado di fare previsioni per periodi estremamente lunghi, fino a 5376 valori in alcuni casi. Abbiamo dimostrato che anche per periodi così lunghi i risultati sono buoni se l'insieme dei dati ha una periodicità costante e regolare. Da un punto di vista interpretativo, questo significa che se c'è regolarità nei dati, i modelli di rete neurale sono in grado di adattarsi, riuscendo a ottenere le informazioni necessarie quando i valori sono periodici. I modelli da noi proposti, realizzati con livelli di complessità sostanzialmente ridotti rispetto ad altri studi, raggiungono prestazioni comparabili e in molti casi superiori rispetto allo stato dell'arte. Abbiamo anche dimostrato che se la condizione di regolarità non esiste, le previsioni a lungo termine sono più problematiche e spesso impossibili.

Parole chiave: Machine Learning, Deep learning, Time Series, Transformers, Neural Network, Previsioni

1. Introduction

It can be assumed that various phenomena occurring in nature or caused by human technology can be represented by numerical time series. In order to predict these phenomena, it is necessary to find models capable of receiving sequences of data as input and calculating numerical outputs that represent the phenomena at successive times. The ambition is to predict the unknown future from the known past.
It can be seen that the majority of time series are stochastic and it is therefore impossible to make error-free predictions. However, some models studied here achieve acceptable approximations for some series. One characteristic of time series is that data have a natural temporal ordering and this is certainly a very important feature that is exploited in deep learning models. In this sense, time series are sequences of vectors in which each of them is coupled to a time value that has certain meaning. However, in studies, this meaning is hardly ever used in calculations and this is the case in this work. This means that what matters is only the order of the values and not the time value to which each belongs. As mentioned, however, the order does have a precise meaning and from this point of view it can be assumed that deep learning models attempt to calculate correlations between different values by means of parameters, which is often greater between closer values than between more distant ones [15].
As a preliminary remark, it is noted that any model that has the ability to predict the future must receive from the known past the information necessary to know it. The known past must therefore have information about its unknown future. The values that make up the time series must contain this information that can be exploited by the models. It follows that if there is no information about the future in the past, no method can obviously find any patterns. This is the case, for example, with completely random phenomena for which time series techniques can do nothing to predict the future. If, on the other hand, there were patterns in the past, it is possible that these would be repeated in the future. In this case, the model must extract this information and then process it in order to function.
There are many active machine learning research fields that study new models or refine existing ones [11]. The Deep Learning models we are dealing with are certainly an important part of all existing ones. There are numerous active lines of research dealing with them using different techniques (a survey: [1]). A significant proportion of the more recent ones use models derived from Transformer (a survey: [13]), which has proven to work quite well in computer vision (a survey: [12]), linguistic analysis (a survey: [7]) and more. Many researchers have therefore found it natural to try to extend its use to other areas and in particular to time series in order to predict future trends. The related techniques have certain characteristics. Firstly, they have a training phase in which the models are found. For Deep Learning models to work, it is necessary that a parametric function representing the evolution of the phenomenon as a function of time is found in the training phase, particularly during back propagation. This function, also known as the optimisation function, must obviously be able to calculate the representation of the phenomenon that minimises the difference between the known reality and the result of the function. The model is successful if the function found is able to calculate the phenomenon with sufficient approximation even when the input is not part of those used during training. Unfortunately on a theoretical, i.e. mathematical, level, the minimisation of the function does not guarantee success. Almost always these functions are not convex and there is very little guarantee that it is possible to solve the problem analytically. In fact, the solution of the equation of minima is not analytical, but empirical and is obtained by successive approximations.
Deep learning models normally act blindly on phenomena. This happens because the model built after the training phase, formed by the weights and hyper-parameters, has adapted itself transparently to the time series itself. In the found and winning model, there is no knowledge of the reasons why the model works. All that is known are the values of the hyperparameters that have the necessary elements of the model in them, but the physical reasons remain hidden in the numbers, which often number in the millions. It is worth noting that the question of how and why the neural network is able to concretely predict a given phenomenon is a active field of research, so that researchers working in the scientific field through Deep Learning techniques also understand the mechanisms that drive phenomena [11].
This work aims to simplify the techniques currently in use to see if this can lead to improvements in terms of both complexity and performance.

2. Releted works

The performance improvement, understood as the ability to predict more exact and longer unknown time sequences, of Deep Learning models applied to Time Series is an active research field. There has been in very recent times and within a short time many researches which have reported an improvement in the state of the art [14]. In particular, different variants of the Transformer [13] have affected different research groups. For

example: FEDformer [20], Autoformer [16], Informer [19], Pyraformer [9], LogTrans [8] , and Reformer [6]. The Transformer was born to solve Natural Language Processing (NLP) problems. Note that the inputs of the NLP models are words that are converted into numbers through embedding techniques. In this sense, the NLP inputs are sequences data. Time series are also sequence data with the added feature of having time as an input data. This important similarity - both are sequence data - makes Transformer mathematics usable for time series problems, since in both cases the input is a numerical vector. However, the original Transformer requires significant adaptations because the scope of the NLP and Time Series study problems are different. Below we will indicate some features and in some cases the necessary adaptations.

1. The inputs of an NLP problem are sentences that through pre-processing become numerical vectors whose values are the function of a pointer to a dictionary. In contrast, the inputs of a time series problem are directly numerical vectors that are not pointers. It should be noted that the two pre-processing steps are also different because in the case of NLP, a dictionary is involved, whereas in Time Series, the dictionary is not necessary.
2. Models that process time series usually belong to the real number domain. It is also a consequence of the standardization process subsection 3.1 which during the preprocessing phase converts the input into as many real values. Since the Transformer has the constraint that the input must be made up of vectors or matrices of fixed length, it is essential to divide the inputs into windows of equal length in a conventional way (see subsection 3.2).
3. The transformer has multiple applications in NLP problems because it can generate texts, translate, and so on. To perform these functions, the transformer in NLP problems normally has a final layer softmax which is used to obtain a classification. In the case of Time Series, on the other hand, the output is a vector or matrix of continuous values that never represents a classification. Consequently in Time Series, the output is never a one-hot vector that, through a softmax activation function, identifies a dictionary word as the most probable. Instead, the ultimate goal in Time Series problems is always to find a vector of numerical values that are never a probability distribution, and consequently the final softmax layer is completely useless.

The core layer of Transformer is the Attention Layer that calculates the similarity matrices of 2 completely connected layers that join its softmax with a third matrix. All 3 matrices have the same input consisting of the Windows input plus the embedded vector, but have a different weight matrix associated with it.
In the original transformer those 3 matrices are called Q(ueries), K(eys), V(alues).
More in details the similarity is calculated from the inner product $< \mathrm{Q} , \mathrm{K} >$ divided by a factor proportional to the input size for scalar reason. Finally transformer apply the filter to the matrix V which has not undergone any transformations except the passage from a Deep Neural Network and therefore can be considered as the original data:

$$\text{Attention} = Softmax\left(\frac{Q\,K^T}{\sqrt{dim_{Q,K,V}=10}}\right)V$$

Transformer-type models have quadratic complexity with respect to the length of the attention level input matrix. In fact, the attention modules perform a product of three matrices of equal size with each other with quadratic complexity with respect to their size. This is a significant obstacle because as the length of the input increases, spatial and temporal complexities that are practically inaccessible are quickly reached.
Also for this reason most of the research works have in mind the objective of reducing the complexity from the quadratic one to a lower level.
The Informer, for example, [19] is a model that applies a variant of the Transformer to Time Series. The main problem its authors have set themselves is to decrease the complexity from quadratic (O(N*N)) to logarithmic O(N*Log(N)) without, of course, losing prediction capability. To do this, they perform a simplification of the matrices calculated by the Attention layer of the Encoder before these matrices are passed to the Encoder. The simplification is achieved through a selection of the most relevant values that eliminates the less relevant ones from the subsequent accounts, simplifying the process. Through a (complex) demonstration, the authors intend to show that the final complexity is precisely logarithmic O(N*Log(N)). However, in the works on the Time Series the general structure of the original transformer with a high number of encoding and decoding modules, put in series, is generally maintained with the consequent spatial (memory required) and computational (time taken) complexity. The Figure 1 shows the the complexity of the classic Transformer model that with the mechanism called multi-head adds Attention layer to the Encoder and Decoder.
The dimensions d-model (the size of the embedding) and dff (the size of the output Multilayer Percepron) are generally kept quite high, which in the original model are respectively equal to 512 and 2048. In this work, an attempt will be made to modify the Transformer to fit the Time Series. However, in contrast to the aforementioned work, a radical simplification of complexity will be undertaken through the elimination of repeated Encoder and Decoder layers and a reduction in the number of parameters. It will be shown that with the elimination of the number of modules and a drastic reduction in the number of parameters, the results will

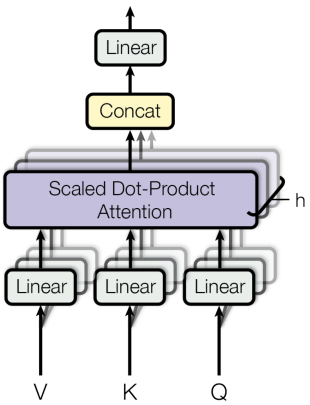


Figure 1: The schema of the Multi-Head Attention mechanism of the vanilla Transformer.

improve instead of deteriorating as might be expected.

3. Models

The task of finding a model capable of guaranteeing forecasts in a generalized way for the Times Series, that is, independently of the dataset, is an active research problem that is not currently solved. The approach to the solution requires great flexibility in choosing the most suitable model.
This work initially examines a first Transformer model adapted to the Time Series. Models are then proposed that simplify this first Transformer-based model by eliminating the Attention Modules, first contained in the Decoder, then also in the Encoder. The models then become much simpler FNN-type neural networks. In the following section 4 the proposed models will be explained in detail. Furthermore, the results of a Persistent model will be shown as a comparison. We understand a Persistent model as a model that uses exactly past values as predictions, without changing them. The Persistent model assumes that the future is equal to the past. In foregoing any processing, the model applies a drastic simplification that naturally cannot satisfy the researcher because the future cannot be assumed to be equal to the past. However, the Persistent model can provide a further element of verification of the models researched. If the results of Persistent models are better than the models that do the calculations, it is likely that these models have problems and could be assumed to be going in the wrong direction. In other words, if deep learning calculations are worse than a pure repetition of the past, it means that they are probably far from capturing the underlying phenomenon. As you will notice, the results obtained after the simplifications do not worsen the results on average. Furthermore, it will be shown how the results have a fairly characteristic trend, but the quality of the results are strongly dependent on the specific dataset examined from time to time.
The pre-processing phase which is common to all the models examined during this work will be described in detail below. The pre-processing phase is necessary in order to provide the Deep learning model with values consistent with the model itself that is capable of processing.

3.1. Standardization

Neural networks converge much faster if the input data is normalized. All input series were normalized by subtracting the population mean from the population standard deviation:

$$x_i = \frac{x_i^d - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

This simple calculation reduces the ranges of features. It is worth noting that the range of normalized results is not between (-1, +1).

3.2. Windows Input

The input during training is a sequence of numerical vectors often called Windows Inputs (an example: [17]). These are Times Series data sequences characterised by a hyper-parameter that defines their length, which is

Table 1: Windows input examples with corresponding related Targets

Inputs	**Targets**
$X_1 = [10, 23, 12, 65]$	$Y_1 = [4, 78, 23]$
$X_2 = [23, 12, 65, 4]$	$Y_2 = [78, 23, 10]$
$X_3 = [12, 65, 4, 78]$	$Y_3 = [23, 10, 27]$
$X_4 = [65, 4, 78, 23]$	$Y_4 = [10, 27, 32]$
$X_5 = [4, 78, 23, 10]$	$Y_5 = [27, 32, 8]$

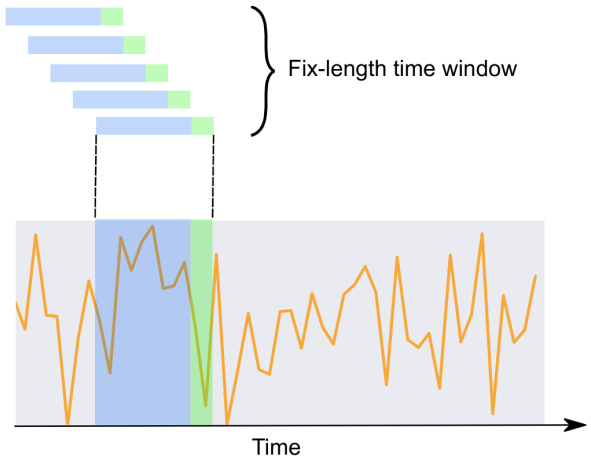


Figure 2: Diagram of how the Windows Inputs follow each other and make up the overall input. This calculation takes place during the pre-processing phase. [17].

fixed. These windows overlap each other with a single time interval to make the best use of the available data. To be more clear Table 1 highlights the vectors of the Windows Input and Target in the case of the Time Series of input x = [10, 23, 12, 65, 4, 78, 23, 10, 27, 32, 8] with the length of the Windows of 4 and Forecasting length of 3.

It is absolutely necessary to use Windows Input because the input Transformers models always have vectors or numeric matrices of a fixed size that are processed one at a time. It is the very logic of the Attention layer that works this way. Its purpose in fact is to calculate the relationships between the different parts of the input vector by marking the most important ones with corresponding higher values of the output matrix. Consequently, the Transformer cannot work continuously by adding input values one at a time, but works 'in blocks' so to speak, i.e. by adding vectors or matrices made up of several values. From a theoretical point of view, the longer the input vector, the better the Transformer would work because it is able to compare and process more data at a time. However, as mentioned, the complexity of the Attention layer is quadratic and its size must therefore be limited taking into account the processing capacity of the hardware. The maximum length also depends on the complexity of the model, but to a lesser extent. For example, adding an Attention layer to an existing one so that the layers become two only doubles the time required, all other conditions being equal. Doubling the size of the input vector, on the other hand, has a quadratic impact on time.

Data windows are computed together with the target vectors during the pre-processing phase Figure 2. These are self-supervised models because the targets are calculated directly and automatically from the input data and not, as is the case with supervised models, through separate work of classification or verification. In supervised models, the targets are in fact obtained through prior work that often involves work by a few people. For this reason, obtaining targets in supervised models is time-consuming and often costly. In any case, it is worth emphasising that the distinction between self-supervised and supervised models does not only apply during the training phase and during the prediction phase. In fact, during training the two models are equivalent. The difference is in how the "target" vectors are calculated [10]. In general, this approach is widely used in models dealing with sequential data and in particular into NLP problems and constitutes a significant advantage because the targets are obtained without particular effort and without human intervention.

Returning to the inputs vectors, also called the Windows Inputs, are processed sequentially and each processing modifies the parameters of the optimization function trying to improve its results. However, each input window represents a different trait in the input sequence. This in practice means that the values of the different traits

considered by the different input windows can also be very different from each other in both value and pattern. These differences are certainly a problem because the resulting pattern is, on the contrary, unique. That is, it occurs that a single pattern must interpret parts of sequences that may be different from each other. The diversities are proportional to the regularity of the data in the dataset. the less regular the pattern of the dataset, the more different the patterns of the different Windows Input are. From an interpretative point of view, if the different traits contain similar patterns, the model will be able to understand and apply them providing good predictions. If, on the other hand, the patterns are different, but nonetheless the patterns exist, the model will have to store them in the training phase and during the testing phase it will have to recognise them in the different Windows Inputs and will have to know how to apply the correct one. In practice, this is extremely complicated. It is possible, again from a theoretical point of view, that complex models can successfully consider different patterns located in different parts of the Time Series. However, this research has no clear evidence of this. In fact this work goes in the direction of simplifying the models rather than complicating them. This simplification is effective, as will be shown during the presentation of the results, quite well when the dataset is regular, as might be expected. It will also be shown that these windows have an optimal length which depends on the specific dataset and that if the dataset is quite regular their length, excluding certain extremes, does not greatly influence the results. Probably in this case, remaining within an adequate range, the model adapts to the applied length. However, it is noted that the optimal length tends to have an empirical relationship both with the length of the models and with the periodicity of the dataset. This is perfectly plausible. The optimal length was found by applying a grid pattern by examining the input windows of multiple length of the periodicity (see subsection 6.4).

3.3. Positional embedding

Before calculating the Q, K, V matrices, all Transformer models, both those applied to NLP problems and those applied to Time Series, must apply a positional embedding to inject into the model the information relating to the order in which the values are sequenced. This process is critically important with time series because it is clear that the order of values is just as important as the number that represents them and it is worthwhile to delve a little deeper into the subject. For example, consider numerical values placed in two different sequences. In the first, all values are placed in ascending order while in the second, the same values are placed in descending order. The number following each of the two sequences will be completely different in the two cases, but the only thing that distinguishes the two sequences is the relative position of the values. In the case of Transformer models applied to time series, there is an absolute necessity for the models interpreting them to be able to take the position of the values into account. It is worth pointing out because it makes a difference with NLP ploblems. In fact, in NLP ploblems, the inversion of some parts of the text does not always change the translation or meaning. In NPL, therefore, there is greater flexibility and a certain tolerance towards errors in the reconstruction of the sequence.
Unfortunately, the Attention module is unable, without fixing, to distinguish the position of the input values. This fix includes a vector called positional embedding which is injected, in various ways and solutions, to the input vector.
Mathematically, the Transformer Attention module is Permutation Invariant that in this context refers to the fact that the model does not assume any spatial relationships between the features. The concept is illustrated in Figure 3. This means that the order in which the values make up the input vectors is indifferent to the model. This represents a serious problem for the Transformer applied to time series because the transformer-type models process the Input vectors regardless of the position of each numerical element while during the analysis of the Time Series it is essential to take them into account.
Regarding the possible solutions, some observations can be made preliminarily.
First, the simpler idea of putting the positions of the tokens in the vector does not work because with long sequences the value of the vector grows too much as illustred in Figure 4.
The idea of using normalized positions doesn't work either, as illustred in Figure 5, because:

- same positions in sequences of different lengths have different values
- long sequences can have minimal differences

The solution devised in the original transformer is function using sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/dmodel})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/dmodel})$$

The authors report:

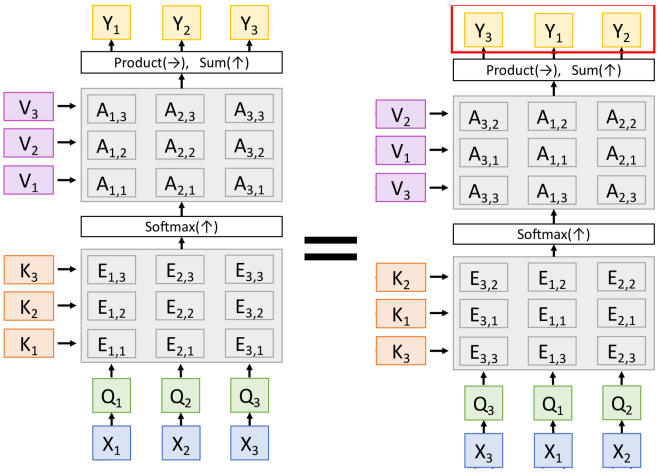


Figure 3: [4] diagram showing how input sequences that have the same values in different positions nevertheless have identical results.

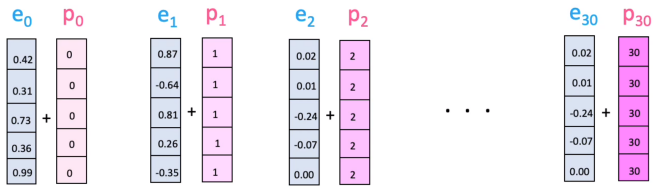


Figure 4: Diagram showing that inserting the token positions in the vector does not work because it causes the values to grow excessively.

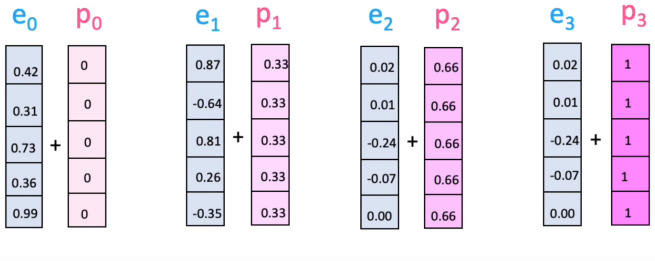


Figure 5: Diagram showing that the use of normalized positions does not work because long sequences have minimal differences and furthermore the position depends on the length of the sequence.

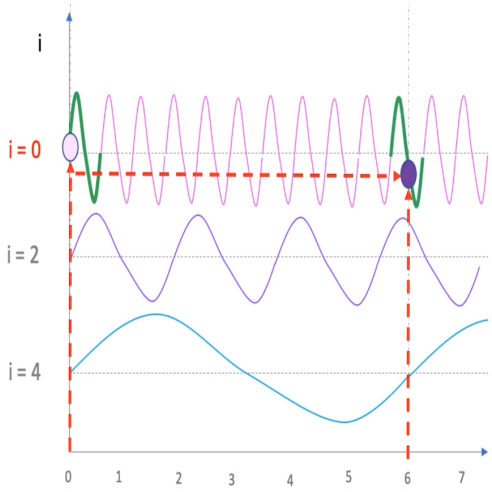


Figure 7: Architecture of the Transformer with shows embedding with sin/cos function. Dot line are 2 examples with position equals 0 and 6 with i equals to 0,2,4.

"That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, $P\ E_{pos+k}$ can be represented as a linear function of $P\ E_{pos}$" [13].

What the authors said is that there is a linear relationship between relative positions in the Transformer's positional encoding and holds:

$$T^{(k)} E_{t,:} = E_{t+k}$$

It can also be shown [2] that the transformation matrix (Figure 6) $T^{(k)}$ does not depend on k and that it probably holds only if the function is indeed sin and cos. In practice, the positional embedding matrix achieves what is desired because:

1. has the same size as the token embedding vector and therefore the two can be added together
2. is limited so the resulting vector cannot "explode"
3. each term depends on both position (pos) and position embedding dimensions (i)

In particular, point (3) has the meaning that each term of each row of the positional-embedding matrix associated with a specific i defines the frequency of the sinusoidal function. Therefore, since the waveform of the function for each i is different, it follows that each term on the same input window slide is necessarily different from another (Figure 7).

$$\boldsymbol{E}_{t,:} := \begin{bmatrix} \sin\left(\frac{t}{f_1}\right) \\ \cos\left(\frac{t}{f_1}\right) \\ \sin\left(\frac{t}{f_2}\right) \\ \cos\left(\frac{t}{f_2}\right) \\ \vdots \\ \sin\left(\frac{t}{f_{\frac{d_{\text{model}}}{2}}}\right) \\ \cos\left(\frac{t}{f_{\frac{d_{\text{model}}}{2}}}\right) \end{bmatrix}$$

$$f_m = \frac{1}{\lambda_m} := 10000^{\frac{2\,i}{d_{\text{model}}}}$$

Figure 6: Trasformation matrix of embedding

3.4. Positional embedding: Time2Vec

In the case of the present work and unlike the classic Transformer and other works, the choice of positional coding is an algorithm obtained from Time2Vec [5] which was devised by its authors with the aim of providing an embedding function positional applicable to different models.
Time2Vec is a learnable vector representation (or embedding) for time as a vector representation. It has features suitable for time series which are:

- capable of capturing periodic and non-periodic patterns
- invariance to time rescaling
- simple to be usable the different models

The i^{th} of the Time Vector of time τ is:

$$\mathbf{t2v}(\tau)[i] = \begin{cases} \omega_i \tau_i + \omega_i & if\ i = 0 \\ F = (\omega_i \tau + \phi_i) & if\ 1 \leq i \leq k \end{cases}$$

where F (according with authors *sin* in the best in order to capture periodic patterns) is a periodic activation function, ω and ϕ are the frequency and the phase-shift and are learnable parameters.
Clearly Time2Vec is inspired by positional-encoding [13] used in Transformer with embedding layer, but:

- represents continuous time instead of discrete time
- is learnable
- is able to capture periodic behaviors which is not the goal in positional encoding
- vector is a model input and is concatenated rather than added to a vector

For the purpose of Neural Networks, Time2Vec is a group of Perceprons layers with *sin* as the activation function and integrates well into the models studied here. In the models proposed here, where it is used, the original vector and the Time2Vec vector are added together (in the original proposal they are concatenated) as in the Residual Network [3] scheme that seems works well. A further disadvantage that would make concatenation really uninteresting is that the size of the input vector doubles with a quadratic impact on complexity. From some tests carried out, however, the sum does not worsen the results compared to the concatenation.

3.5. Output

In all the models examined in this work, including ones based on Transformer, the output is a numeric vector that directly constitutes the forecast and has a variable length between 24 and 5376, therefore with a very long forecast length. Note that in previous works, probably also due to the complexity of the models and the limitations of the hardware available to the researchers, the prediction has a much shorter length, up to 720 values.
The last layer suitable for calculating a vector (or a matrix) of the values that make up the output is Fully Connected Layer made up of Perceprons Layers with or without an activation function (depends on the models).

3.6. Optimization

The output vector is computed from a final linear parameterized layer that directly outputs the prediction vector. During the training phase this vector is compared with the target vector which is exactly the same size. The two vectors are applied to the formula to calculate the Mean Absolute Error (MAE):

$$\sum_{i=1}^{D} |x_i - y_i|$$

Two errors are commonly compared, the MAE and the Mean Squared Error (MSE). However, even looking at the time series works mentioned in this document, MSE does not seem to add useful information in order to find better performing models also because we are evaluating the absolute value of the prediction error. Consequently, to streamline the tables of results, the choice was made to use only the MAE. The MAE compared to the MSE has the advantage of being linear with respect to errors with a consequent more intuitive understanding.

4. Model architectures

Some characteristics regarding the preprocessing, the traning and forecasting phase referred to section 3 are common to all the models examined in this work. The models that have been studied in this work will be compared with the state of the art (autoref sec: results). The same models have some specific characteristics which are explained in the following paragraphs.

4.1. Encoder + Decoder

First model that we have called "Sinformer" derived directly from the Transformer. The structure is the one described in Figure 8. It is simplified compared to the original Transformer because the Encoder and the Decoder are formed by only one layer each unlike the original Transformer in which, through the parallelization of the input sequence through a mechanism called Multi-Head-Attention, it divides the accounts into multiple sections (called Head). Also the dimension of Attention matrices (d-model) and the dimension of final layer of Encoder and Decoder (dff) are limited to 128, while in the original transformer they are respectively 2048 and 512. The final layer (which in the original Transformer is a softmax type layer) is a Full Neural Network that

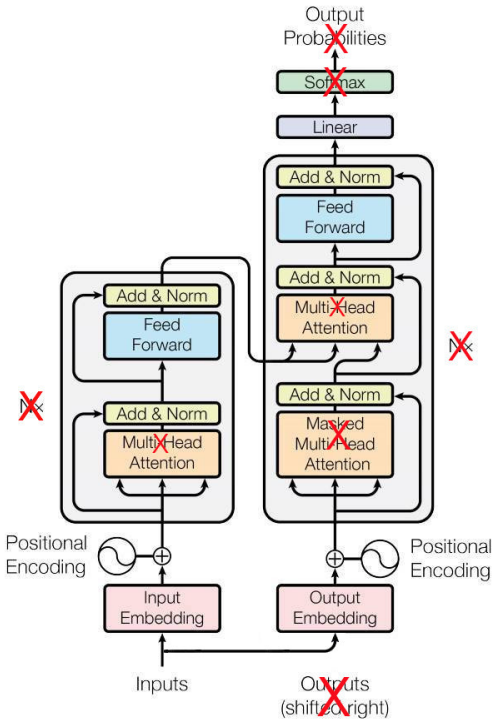


Figure 8: Architecture of the Transformer [13] with indicated the parts deleted for conversion to the architecture of the model. Eliminated parts are marked in the Transformer and are covered with a red X.

allows you to have an output matrix that reproduces the segment of the Time Series object of the forecast. The idea is to terminate the model with a periodic function similar to the one placed at the beginning of the model in which there is Time2Vec.
In the original Transformer applied to NLP problems the output is a numeric vector of the same length as the dictionary and is one-hot which means that its largest value (argmax) points to the word in the dictionary. So the only value that matters is the highest one and all the others are not used. This is not true with Transformer models applied to time series because it is a regression and not a classification problem. This means that the entire output vector is useful because it represents the forecast being sought.

4.2. Encoder

This second model is the same as the previous one but composed only of the decoder part. Only the Encoder module is implemented. The output vector is calculated directly from the Encoder module, which is normally one of the Decoder's inputs. This output vector is compared with the "target" vector. The purpose of the model is to check whether the decoder module adds information capacity to the model and is therefore useful for improving the results. From an interpretative point of view, this is not obvious because in Transformer models applied to Time Series no activation phase is necessary.

4.3. Multilayers

The third model is a Full Neural Network with 3 layers. Each layer has a 'relu' activation function and a final regularization of type L1 and L2. This model with a further simplification compared to the previous model, does not have the Attention module and has 3 linear layers without the Attention layer. The purpose of the model is to verify if the Attention module adds a real information capacity to the Deep learning models applied to the Time Series.

4.4. One layer

It is a neural network with only one layer with a further simplification compared to the previous model. The purpose of this model is to test whether a single linear layer can perform better or worse than more complex networks with multiple layers or Attention Modules. From an interpretative point of view, the answer is by no means obvious and it is only after the results of the experiments are examined that it will be answered.

4.5. Persistent

It is obviously not a Neural Network because predictions are made simply by assuming that the past can repeat itself. The forecast consists of exactly the same number of closest past values. It is used only as a basic comparison to understand the improvements actually made by the Neural Network models. From an interpretative point of view, if the Deep Learning model performs worse than the Persistent model used as a comparative basis, it probably means that the model is not on the right track to solve the problem.

4.6. Liner regression

The Linear Correlation (without penalty) was calculated as a comparison term for each prediction. This is an extremely simple model that differs from Deep Networks. However, it has some interesting aspects specific to Time Series. In fact, since the Time Series models examined in this work all work with Windows Input in order to have a homogeneous comparison, it was necessary to apply the same principle to the linear correlation model examined here. We have therefore redetermined the linear correlation parameters for each Windows Input so that each subsequent forecast can be calculated with an up-to-date model. From an interpretative point of view, it can be assumed that this model performs worse than deep learning models, and indeed, one can anticipate that this is indeed the case.

5. Training

The proposed models are tested on the Time Series shown in this section. Before examining the results of the experiments, we chose to display the autocorrelation graphs as they are representative of periodicity and can intuitively provide the presence or absence of more or less regular periodicities. It will be seen that the more autocorrelation indicates a regular periodicity, the more accurate the predictions are, with particular reference to long-term predictions. Conversely, the lack of periodicity will lead to less accurate forecasts. In the remainder of this work, when the results are presented, evidence will be provided to confirm that these two circumstances are true.

- ETT [21]. These are values of an indicator that measures the distribution of electricity in two Chinese counties that covers the span of two years with 1 hour (ETTh1, ETTh2) and 15 minutes (ETTm1) granularity. Each row of the dataset consists of the "oil temperature" target value (OT column) and 6 power load characteristics. The train/val/test is 12/4/4 months. This dataset actually constitutes a Benchmark because it is widely used to test forecast models based on Neural Network. It certainly has the interesting feature of the periodicity graphically represented through the autocorrelation graphs Table **??**.

  As you can see, the periodicity is marked but not completely regular. In particular HULL, MULL, LUFL, LULL have a clearly decreasing moving average and OT has a moving average which decreases then increases etc ...
  This dataset provides multivariate data.

- **DAX**(Figure 12). Hourly values that is a stock market index consisting of the 40 major German blue chip companies trading on the Frankfurt Stock Exchange. The train/val/test is 90/15/15 days. Note the absence of periodic phenomena in the graph. This dataset provides multivariate data.

- **Venezia**(Figure 15). Sea level in Venice. These are the values (hourly) of the tide level recorded in Venice from 1983 to today. The train / val / test is 12/4/4 months. The periodicity is double every day and the maximum peaks are every 12 hours. In more detail, it can be observed that in each 24-hour period there are two peaks, one of which is lower. Apart from this, the auto-correlation graph shows a

Table 2: ETTh1 auto-correlation - There are the autocorrelation graphs of all the columns of the input matrix. All have a clear periodicity.

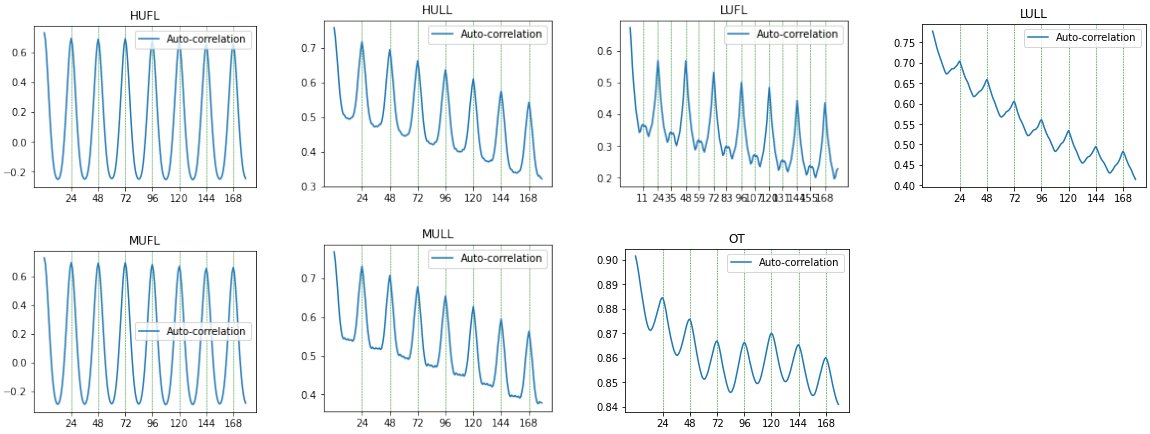


Table 3: ETTh1 auto-correlation - There are the autocorrelation graphs of all the columns of the input matrix. All have a clear periodicity.

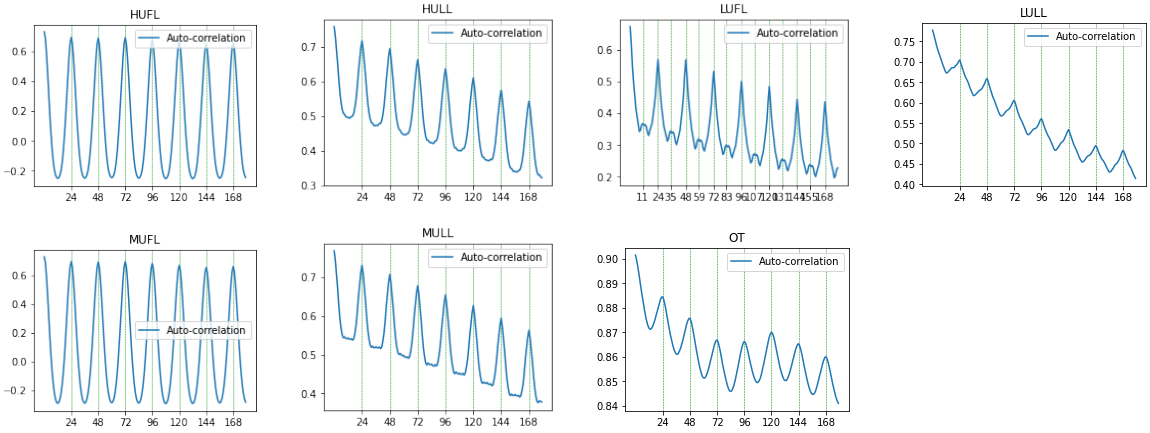
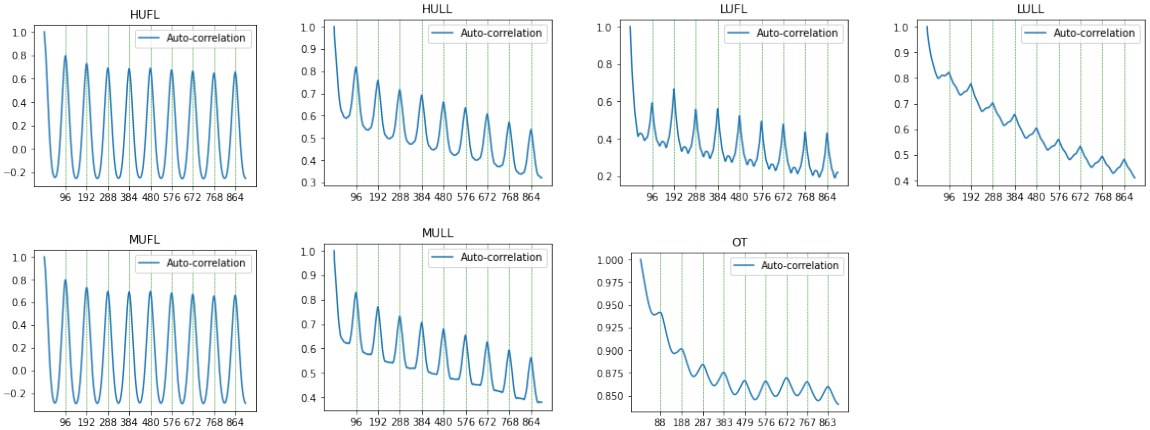


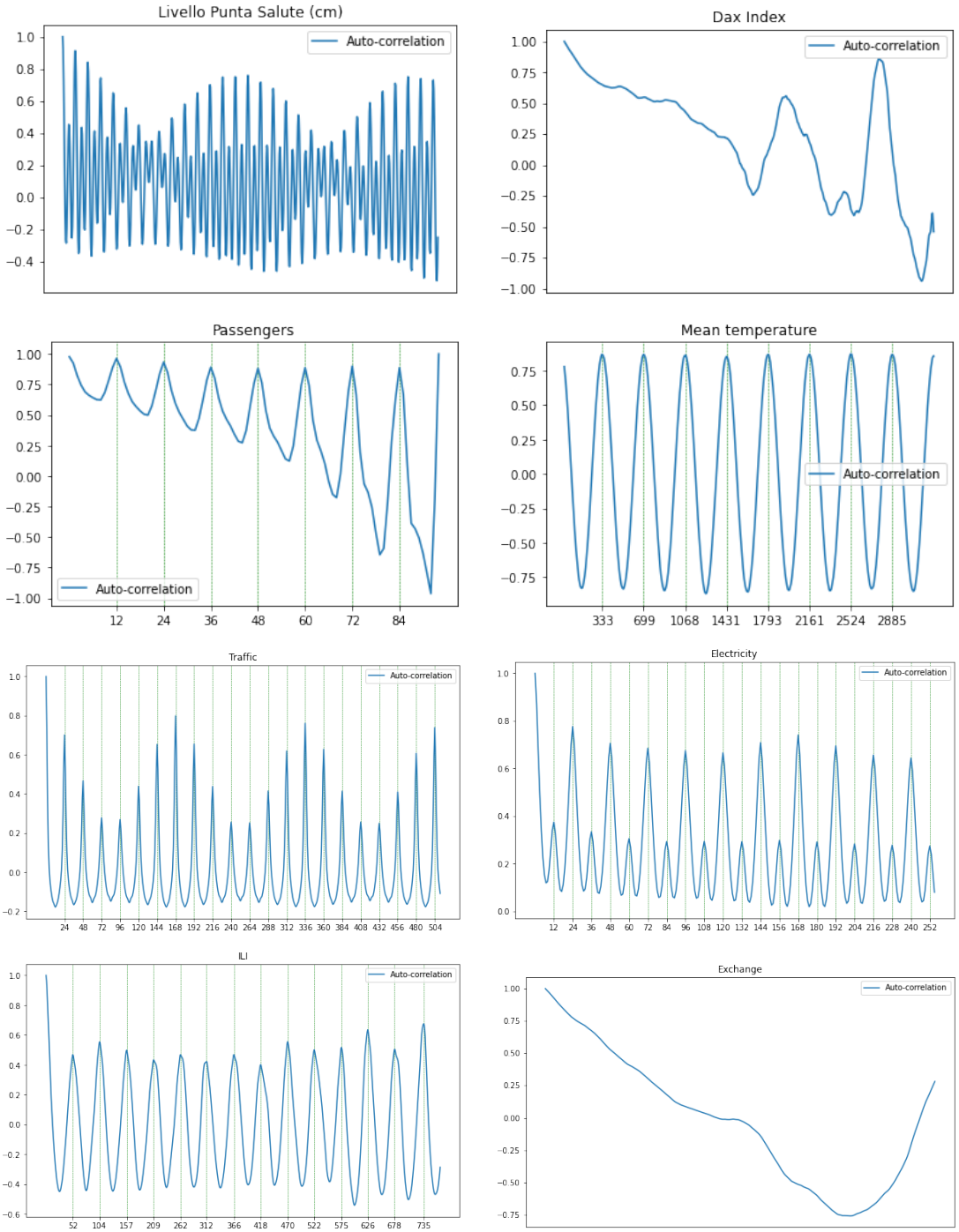
Table 4: ETTm1 auto-correlation - There are the autocorrelation graphs of all the columns of the input matrix. All have a clear periodicity.



strong regularity which has a positive impact on the predictive ability. The predictive model in this case is univariate because there is only one data vector representing the height of the tide.

- **Airlines**. It is a dataset containing the number of scheduled passengers between 1949 and 1960. The periodicity is very evident with a cycle of 12 months. This dataset provides univariate data. They are

Table 5: Group of autocorrelation graphs of the specified datasets.



monthly surveys.

- **Temperature**(Figure 14). Milan temperatures collected daily from 2001 to 2021. The train/val/test is 12/4/4 years. This dataset provides univariate data. The periodicity is clearly annual.

- **Traffic**. Escribes the road occupancy rates. It contains the hourly data recorded by the sensors of San Francisco freeways from 2015 to 2016. This dataset provides multivariate data.

- **Electricity**. Collects the hourly electricity consumption of 321 clients from 2012 to 2014. This dataset provides multivariate data.

- **Exchange-Rate**. Collects the daily exchange rates of 8 countries from 1990 to 2016. This dataset provides multivariate data.

- **ILI**. Describes the ratio of patients seen with influenza-like illness and the total number of the patients. It includes the weekly data from the Centers for Disease Control and Prevention of the United States from 2002 to 2021. This dataset provides multivariate data.

- **Weather**. Includes 21 indicators of weather, such as air temperature, and humidity. Its data is recorded every 10 min for 2020 in Germany. This dataset provides multivariate data.

The training phases were performed by applying two different methods depending on whether it was a multidimensional or one-dimensional study, according to the methodology illustrated below:

- If the input is a multidimensional matrix, the backpropagation step is performed for each column of information for each epoch. This means that as many backpropagation steps are performed for each epoch as there are columns in the input matrix. Each step works on a specific column, changing the value of the weights each time to try to get closer to the optimal values. It has been noted that in this way, at least in the series of experiments carried out in this work, backpropagation works better at finding the optimal parameters than a single backpropagation step done at once for each epoch over the entire table. It should also be noted that working on one column at a time greatly reduces spatial complexity compared to working with the complete input matrix. This is especially useful if the table is made up of many columns (and this is also the case in the datasets studied because there are matrices made up of hundreds of columns) because in this case the spatial complexity with the complete input matrix quickly becomes very high and this means that the hardware-compatible models must have rather short Windows inputs. On the other hand, by working on one column at a time the memory space required drops dramatically and the length of the Windows Input can remain rather long and closer to optimal. However, it must be emphasised that the time complexity, on the other hand, increases a lot because there are many more backpropagation phases. It is true that a single backpropagation step working on a single column is much simpler than a step working on the entire input matrix, but this does not compensate for the large number of backpropagation steps that are required in the former case.

- In the case of one-dimensional dataset the backpropagation phase is performed for each dataset.

Calculations are performed on 50 epochs and the baseline result is considered to be the one in which the forecast on the valuation set is best.

6. Results

The section just concluded was used to choose the models to compare with those at the state of the art. And now we bring the best ones with us and present the various comparisons.
In this section I am going to present the experiments related to the models presented and the datasets previously described. The results of each model will be examined in the subsection 6.1. The general and overall results for specific dataset are reported in following tables:

- For ETT datasets are reported in Table 6.

- For DAX dataset are reported in Table 7.

- For Bitcoin dataset are reported in Table 8.

- For Venezia dataset are reported in Table 9.

- For Airlines dataset are reported in Table 10.

- For Temperature dataset are reported in Table 11.

Before presenting the results of the different models, however, a general premise must be made. It will be noted that in some tables that intend to compare different aspects of the problem, values referring to the same dataset and with the same hyperparameters are repeated, but compared with the results of tests made with different models or with different datasets. Despite the similarity, the values in them may be slightly different even for the same dataset and forecast. This happens because in the different tables, the training phases are repeated

Table 6: Multivariate ETT MAE - Bold shows the best results for each row

Dataset	Forecast	Linear	Persistent	One layer	Encoder	Multilayers	Encoder + Decoder
ETTh1	24	0.767	0.386	0.34	0.365	**0.338**	0.353
	48	2.043	0.423	0.362	0.366	**0.361**	0.381
	168	0.697	0.524	0.419	0.427	**0.418**	0.433
	336	0.741	0.571	**0.462**	0.473	0.472	0.489
	720	0.865	0.718	**0.533**	0.554	0.557	0.581
	1344	1.001	1.291	**0.622**	0.667	0.659	0.675
	2688	1.574	1.601	**0.781**	0.825	0.843	0.905
ETTh2	24	0.524	0.305	**0.265**	0.279	0.274	0.27
	48	0.59	0.373	**0.306**	0.332	0.33	0.317
	168	0.657	0.486	**0.4**	0.468	0.459	0.417
	336	1.102	0.56	**0.471**	0.533	0.56	0.518
	720	1.281	0.682	**0.577**	0.714	0.696	0.754
	1344	1.564	1.711	**0.626**	0.721	0.783	0.698
	2688	6.247	2.371	2.51	2.51	**2.06**	2.191
ETTm1	24	1.804	0.727	**0.283**	0.314	0.296	0.309
	48	0.945	0.826	**0.322**	0.379	0.329	0.376
	96	0.828	0.389	**0.34**	0.367	0.343	0.387
	288	0.758	0.443	**0.377**	0.408	0.38	0.438
	672	0.721	0.52	**0.416**	0.471	0.453	0.486
	1344	0.894	0.619	**0.492**	0.527	0.51	0.528
	2688	0.98	0.803	**0.568**	0.594	0.592	0.793
	5376	1.282	1.277	0.657	**0.652**	0.669	0.682

Table 7: Multivariate DAX MAE - Bold shows the best results for each row

Dataset	Forecast	Linear	Persistent	One layer	Encoder	Multilayers	Encoder + Decoder
dax	24	0.213	0.108	**0.091**	0.235	0.288	0.224
	48	0.219	0.18	**0.131**	0.183	0.236	0.298
	96	0.275	0.276	**0.185**	0.237	0.246	0.286
	288	0.469	0.403	0.315	0.298	**0.272**	0.505
	672	0.708	0.685	**0.366**	0.504	0.41	0.487
	1344	2.298	1.198	**0.369**	5.805	5.178	5.734
	2688	–	2.917	5.144	4.44	4.746	5.266

with new calculations and it must be clear that each time the training phase is repeated, the results are always slightly different while remaining similar due to non-reproducibility due to GPU computation.
One of the problems encountered in much of the work done by researchers on Transformer models has been coping with complexity that is quadratic with respect to the size of the input vector and thus reaches levels that are impossible to handle from the standpoint of available hardware even with window sizes of a few hundred values per row. The same problem was examined in this work specifically by simplifying the Transformer model in the number of parameters and then abandoning the Transformer (section 4). One might have expected different results depending on the degree of simplification, but this does not seem to be the case. In fact, the results show that for different datasets all the neural networks examined in this work do not perform very differently

Table 8: Multivariate Bitcoin MAE - Bold shows the best results for each row

Dataset	Forecast	Linear	Persistent	One layer	Encoder	Multilayers	Encoder + Decoder
bitcoin	24	0.805	0.436	**0.384**	4.14	7.133	7.072
	48	1.044	0.583	**0.492**	3.759	6.595	7.002
	96	1.203	0.846	**0.681**	4.442	6.77	6.94
	288	**1.314**	1.363	1.787	4.027	5.8	6.377
	672	**1.573**	1.663	3.858	4.75	4.399	5.666
	1344	**1.484**	1.905	8.589	3.475	3.063	4.146
	2688	**0.753**	1.846	20.587	2.221	1.507	2.244

Table 9: Univariate Venice MAE - Bold shows the best results for each row

Dataset	Forecast	Linear	Persistent	One layer	Encoder	Multilayers	Encoder + Decoder
venezia	24	1.034	0.333	**0.185**	**0.185**	0.188	**0.185**
	48	0.991	0.568	**0.225**	0.229	0.233	0.227
	96	0.882	0.936	**0.265**	0.282	0.287	0.279
	288	0.965	0.772	**0.313**	0.348	0.385	0.343
	672	1.02	0.618	**0.34**	0.372	0.405	0.362
	1344	0.886	0.891	**0.352**	0.391	0.516	0.376
	2688	0.838	1.307	**0.358**	0.419	0.454	0.403
	5376	0.826	1.017	**0.354**	0.402	0.439	0.387

Table 10: Univariate Airlines Passangers MAE - Bold shows the best results for each row

Dataset	Forecast	Linear	Persistent	One layer	Encoder	Multilayers	Encoder + Decoder
airlines	6	2.028	2.225	**1.512**	5.044	5.093	5.584
	12	2.526	**1.639**	3.045	6.983	6.536	7.245
	24	**0.941**	3.792	5.829	9.723	9.279	10.543

Table 11: Univariate Milan Temperature MAE - Bold shows the best results for each row

Dataset	Forecast	Linear	Persistent	One layer	Encoder	Multilayers	Encoder + Decoder
temperature	28	0.669	0.54	0.334	0.436	0.396	**0.291**
	56	0.991	0.91	0.347	0.4	0.378	**0.316**
	224	0.929	1.663	0.347	0.361	0.369	**0.345**
	364	0.422	0.464	0.346	0.353	0.378	**0.337**
	728	0.51	0.386	**0.358**	0.373	0.407	**0.358**
	1456	–	0.43	0.384	0.363	0.388	0.356

from each other. This is generally true even for very long prediction lengths, up to 5376 values.
In this work, special attention has been paid to very long-term forecasts by exploiting the reduced complexity of the trailed models. The results are explained to **??**.

6.1. Models in details

Here are the specific observations relating to the results obtained with the various models.

6.1.1 Transformer

The results of the simplified Transformer called "Encoder + Decoder" are comparable to those of the simple neural network, but with a much higher spatial and temporal complexity that makes them much less efficient. A similar conclusion that considers Transformer-based models of little use for analyzing Time Series was already reached in [18]. They test and compare FEDformer [20], Autoformer [16], Informer [19], Pyraformer [9], LogTrans [8] , and Reformer [6]. Each of these models has considerable complexity, although some authors explicitly state that they have worked to reduce it. The realities and models of Transformer remain quite complex. However, the results are worse than the simple model proposed here, as reported in Table 6.

6.1.2 Encoder

This model is the same as before, adapted for not having the decoder. It can be seen by examining the results that models without the Decoder perform very similarly to those without. Removing the Decoder module obviously does not change the quality of the results. The interpretative conclusion is that the Decoder module obviously does not add its own ability to improve the information.

6.1.3 Multilayers

With this model in which the Attention modules are removed altogether, the results do not change substantially. Therefore, the results show that it is the entire Attention module that does not add predictive capability for all the datasets examined. This is quite evident when looking at the results and is certainly surprising.

6.1.4 One layer

Looking at the results, this simple single-layer neural network often obtains slightly better results than more complex models. The difference is often minimal and may mean that the models are quite similar in their results. This is certainly surprising. From an interpretative point of view, the fact that simple neural networks perform as well as, or even better than, more complex neural networks may lead to the uncomfortable conclusion that neural networks are unable to interpret the complexity of time series datasets. In fact, if the opposite were true, a higher complexity of the input dataset should correspond to a higher complexity of the model for good prediction, as happens in image analysis or with NLP problems. But for Time Series this does not happen. Instead, the exact opposite happens, namely that extremely simple networks perform better, and this regardless of the complexity of the dataset. The work of increasing the complexity of models for Time Series to improve results apparently does not work. This observation leaves a future line of research open and it is clear that there is much work to be done to find an eventual solution in this field.

6.1.5 Persistent

It can also be observed (Table 16) that the Persistent forecasting model which has no hyperparameters and which acts as a reference has, in some cases, results comparable to the trailed models studied here, but better than the results obtained by the Informer [19], LogTrans [8]. The approximation with which the Persistent model approaches the exact values is not uniform and again depends on the regularity of the dataset. The conclusion that the Persist model does better than the Informer model (and LogTrans) is surprising considering the efforts made to make such complex models work. But nevertheless, it must be made clear that if a model performs less well than simply using the past as a prediction of the future, it is evident that the model does not capture the patterns present in the input datasets and is therefore absolutely useless from the point of view of improving the state of the art. The only thing that all these experiments show is that some complex models based on the Transformer whose results are far from the Persistent model do not work at all with Time Series.

6.1.6 Linear

The linear model clearly fails and its predictions are generally worse than Deep learning models, with the curious exception of the bitcoin dataset for long periods (over 288 values) multivariate-bitcoin. In the apparent randomness with which this index moves there is a curious ability of a pure linear model to do better than complex Deep Learning models. However, it may be that this result is only the result of a statistical fluctuation and is not repeated for other traits in the related Dataset. Even for the airlines dataset on 24-period surveys, which corresponds to two years because the values have monthly periodicity, the linear model does better than the others.

6.2. General comment on the results

From the interpretative point of view, some observations are possible which have a certain general validity and which are reported below:

1. The fact that similar results are obtained with neural networks of such varying complexity is only possible if the different networks are able to adapt with equal precision to the objective of minimising the function that determines the MAE. Consequently, the complexity of more structured networks is unnecessary and perhaps even counterproductive because, even if only slightly, linear networks without Attention Modules obtain the best results for most predictions.
2. It is possible to have forecasts with a good MAE even for very long periods
3. It can also be observed (Table 16) that the Persistent forecasting model which has no hyperparameters and which acts as a reference has, in some cases, results comparable to the trailed models studied here, but better than the results obtained by the Informer [19], LogTrans [8]. The approximation with which the Persistent model approaches the exact values is not uniform and again depends on the regularity of the dataset. The conclusion that the Persist model does better than the Informer model (and LogTrans) is surprising considering the efforts made to make such complex models work. But nevertheless, it must be made clear that if a model performs less well than simply using the past as a prediction of the future, it is evident that the model does not capture the patterns present in the input datasets and is therefore absolutely useless from the point of view of improving the state of the art. The only thing that all these experiments show is that some complex models based on the Transformer whose results are far from the Persistent model do not work at all with Time Series.
4. The linear regression model, on the other hand, generally performs significantly worse than the neural network models and also the persistent reference. From an interpretative point of view, it can therefore be concluded that the trend all datasets are not linear.
5. Observing all the data, it is evident that an extreme and very general conclusion leads us to conclude that, whatever the model, the goodness of the results is strongly dependent on the dataset and also on the segment of each dataset considered that may or may not have significant patterns.

6.3. Forecast Length

In this section we will analyze the results from the point of view of the length of the predictions.
The results are consistent with the premises highlighted in this work, particular with regard to the effects of the presence of strong regularity and evident periodicity in the datasets. But before presenting the results, it is useful to answer a question. The question arises as to whether these models retain predictive capability even for very long periods, i.e. longer than 720 values, which is the length generally tested in the published research already mentioned. Given the simplicity of the proposed models, the memory of the GPU was sufficient to train the models for very long periods, even 5376 values when the amount of data in the dataset permitted this. The durations correspond to different actual periods, which we report in the table Table 12 table.
Returning to the general results, it is possible to note that the trend is that in which the autocorrelation graphs show regularity in the periodicity, the models maintain very long-term forecasting capabilities, with minimal degradation.
An example of a dataset that allows all models to maintain its forecasting capacity is that of the high water of Venice, studied as a univariate. It was observed that for 24 values (one day) the mean MAE is 0.185, while for 5376 values (32 weeks), it is 0.354. Look at the graphs containing the real data and the predictions of the test set for this dataset reported in the appendix in the figure 15. For this dataset also the lines representing the length of the longest predictions pass very close to the lines of the real values. Here the model is able to reproduce the real function with great precision. It should be noted that the graph is constructed with Window Inputs with

Table 12: Maximum period trained

Dataset	Nr. values	Weeks	Days	Interval
ETTh1	2688	16	112	60 minutes
ETTh2	2688	16	112	60 minutes
ETTm1	5376	8	56	15 minutes
Venezia	5376	32	224	60 minutes
Temperature	1456	208	1456	1 day
Dax	2688	–	–	15 minutes
Bitcoin	2688	–	–	15 minutes
Traffic	720	4	30	1 hour
Electricity	720	4	30	1 hour
Exchange-Rate	720	103	720	1 day
Weather	720	1	5	10 minutes
ILI	720	720	5040	1 week

a length of 1024 values. However, in the **??**, which will be presented in greater detail in subsection 6.4, it is noted that a good predictive capacity is also obtained with much shorter Windows Lengths. Even with lengths equal to 48 hours (two days) the predictive ability is good.
From an interpretative point of view, it can be understood that the regularity of the dataset relating to high water in Venice is so strong that a Windows Input window of only 256 values can predict a time horizon of 2688 values. This is evidently only possible in the presence of very pronounced periodicity. Opposite cases are Dax (Figure 12), where for 24 values the MAE is 0.091 while for 2688 values the best MAE is equal to 4.44 which means a null forecasting capacity, and Bitcoin (Figure 13). Looking at the graphs it is evident that the models applied to Dax and Bitcoin have no forecasting capacity that can be of any use. Notice how the lines have separate courses. The other datasets (Figures 9, 10, 11, Figure 14) have intermediate results. Observing the graphs you have an intuitive perception.

6.4. Windows Input Length

The length of each Windows input is an obviously necessary hyper-parameter in all the models proposed here. The question arises as to whether this hyperparameter affects the results and, more importantly, how much it affects them. To find out, we trained some datasets, namely ETTh1, ETTm1 and Venice, with the 'One Layer' model and with different lengths of Windows Inputs from 48 to 3072. The Table 13 shows the results for the ETTh1 dataset, Table 14 for ETTm1 and Table 15 for Venice. These tables cross the results for the length of the forecast periods with those for the length of the Windows Inputs.

Table 13: ETTh1 Results depending on Windows Input Size- Bold shows the best results for each row

	Windows Size												
Forecast	**48**	**72**	**96**	**128**	**256**	**384**	**512**	**768**	**1024**	**1280**	**1512**	**2048**	**3072**
24	0.34	**0.338**	0.34	0.34	0.342	0.353	0.362	0.399	0.392	0.418	0.529	0.655	0.552
48	0.363	**0.361**	**0.361**	**0.361**	0.364	0.375	0.385	0.4	0.418	0.45	0.62	0.72	0.588
168	0.419	0.423	0.419	**0.416**	0.419	0.433	0.444	0.458	0.478	0.511	0.83	0.85	0.679
336	0.475	0.463	0.458	**0.456**	0.462	0.472	0.477	0.494	0.518	0.867	0.893	0.852	0.74
720	0.547	0.537	0.533	**0.528**	0.531	0.539	0.543	0.565	0.592	0.979	0.924	0.795	0.838
1344	0.632	0.63	0.62	0.657	**0.616**	0.62	0.625	0.644	0.667	0.683	0.711	0.774	0.748
2688	0.779	**0.775**	0.778	0.777	0.796	0.812	0.842	0.909	0.929	0.927	0.91	0.955	1.539

From an interpretative point of view, the length of the Windows Input determines the number of past values that are used for predictions. Ideally, it would be optimal to consider a number of values such that every useful piece of information that can be used by the model's Deep network is included. In practice, it can be observed that results improve as the length of Windows Input increases, but only up to a certain size, and then deteriorate. The conclusion is that the information immediately preceding the prediction is only useful up to a

Table 14: ETTm1 Results depending on Windows Input Size- Bold shows the best results for each row

	Windows Size												
Forecast	**48**	**72**	**96**	**128**	**256**	**384**	**512**	**768**	**1024**	**1280**	**1512**	**2048**	**3072**
24	0.364	0.329	0.3	0.296	0.276	0.269	0.267	**0.265**	**0.265**	**0.265**	**0.265**	0.266	0.268
48	0.418	0.362	0.347	0.345	0.322	0.316	0.314	0.312	0.312	**0.311**	**0.311**	0.313	0.315
96	0.409	0.382	0.372	0.363	0.348	0.343	0.342	**0.34**	**0.34**	**0.34**	**0.34**	0.342	0.344
288	0.456	0.426	0.415	0.408	0.393	0.388	0.386	0.389	**0.385**	0.386	0.386	0.391	0.396
672	0.494	0.465	0.455	0.448	0.434	0.429	0.426	0.426	**0.425**	0.426	0.428	0.433	0.444
1344	0.527	0.501	0.492	0.486	0.472	0.467	0.465	**0.464**	0.465	0.467	0.469	0.475	0.487
2688	0.579	0.558	0.551	0.546	0.535	0.529	**0.527**	0.529	0.53	0.53	0.531	0.531	0.536
5376	0.637	0.622	0.616	0.612	0.602	0.599	0.597	0.596	0.596	0.595	0.594	0.596	

Table 15: Venezia Results depending on Windows Input Size- Bold shows the best results for each row

	Windows Size												
Forecast	**48**	**72**	**96**	**128**	**256**	**384**	**512**	**768**	**1024**	**1280**	**1512**	**2048**	**3072**
24	0.21	0.203	0.199	0.198	0.187	0.185	0.185	**0.184**	0.185	0.189	0.193	0.2	0.204
48	0.268	0.256	0.25	0.247	0.228	0.226	0.226	**0.222**	0.223	0.227	0.229	0.236	0.239
96	0.345	0.324	0.314	0.306	0.273	0.27	0.268	**0.262**	**0.262**	0.267	0.269	0.275	0.279
288	0.443	0.41	0.391	0.373	0.333	0.326	0.318	0.31	**0.309**	0.313	0.315	0.321	0.326
672	0.46	0.43	0.412	0.395	0.357	0.351	0.346	**0.337**	0.34	0.34	0.342	0.344	0.34
1344	0.475	0.446	0.43	0.415	0.385	0.38	0.377	0.366	0.36	0.356	0.354	0.352	**0.339**
2688	0.504	0.48	0.466	0.452	0.424	0.417	0.408	0.386	0.373	0.364	0.357	**0.346**	0.352
5376	0.487	0.461	0.446	0.431	0.4	0.392	0.384	0.367	0.357	0.353	**0.351**	**0.351**	0.359

certain value. This could actually be due to the fact that information that is too old does not add any useful information, but also to the fact that the models do not have the ability to exploit information that is too 'old' by integrating it with newer information. In general, it can be seen that the length of Windows Inputs tends to be a hyper-parameter that depends specifically on the dataset.

7. Comparison with the State of the Art and Discussion

This chapter compares the best results obtained with the final state of the art, to our knowledge, probably constituted by experiments of a model similar to One Layer . This work ([18]) was published during the writing of the thesis (similar in intuition, but at the same time very different in implementation). For completeness, we include a description and a comparison with the results obtained.

In the paper the authors invent a new model that eliminates attention modules. Specifically, the authors compare three linear models that they call Linear, NLinear and DLinear. Specifically, Linear is a rather simple linear model, while DLinear and NLinear are its two variants with two different preprocessing steps. In the first, in DLInear, there is a combination of a decomposition scheme used in the linear Autoenformer ([16]) and FEDformer ([20]). In the second, they apply a calculation to subtract the final value from the whole dataset before applying traning. It does not appear that this method adds anything new because normalisation achieves exactly the same objective.

In the work just cited, there is probably too much emphasis on emphasising small differences in the results. Despite the attempt to differentiate the pre-processing steps, the results are practically the same in all three models and differentiating the pre-processing steps as has been done does not seem to add anything significant to the research. In fact, the differences in the results are in the second and third decimal places of the MAE and cannot really make it possible to construct a meaningful comparison ranking of the goodness of the different models.

We asked ourselves whether the results achieved in this work are different/better than those obtained with Linear/NLinear/DLinear. The comparison is given in Table 16 which shows, for all the datasets mentioned, the

Table 16: Multivariate NLinear MAE - Bold shows the best results for each row

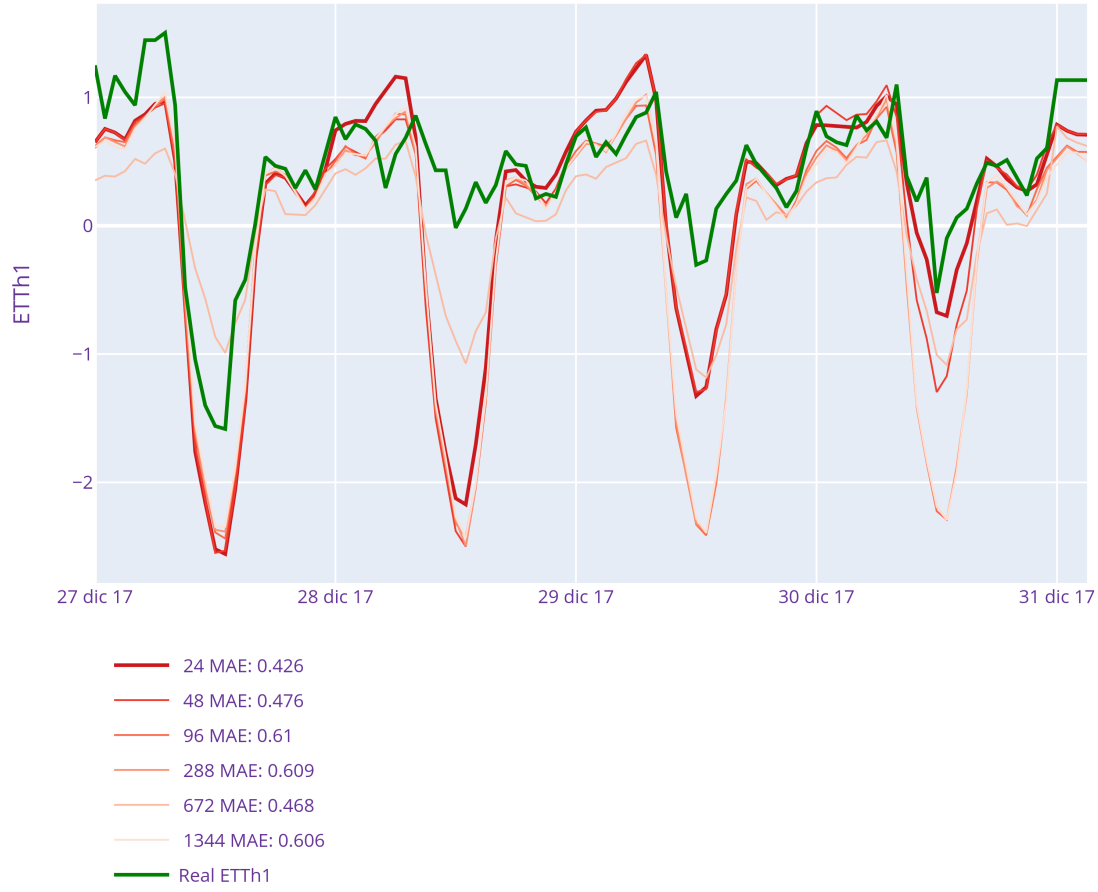
Dataset	Forecast	Persistent	One Layer	NLinear	FEDFormer	Autoformer	Informer	LogTrans
ETTh1	96	0.48	**0.392**	0.394	0.419	0.459	0.713	0.74
	192	0.53	0.423	**0.415**	0.448	0.482	0.792	0.824
	336	0.571	0.456	**0.427**	0.465	0.496	0.809	0.932
	720	0.718	0.528	**0.453**	0.507	0.512	0.865	0.852
ETTh2	96	0.428	0.348	**0.338**	0.388	0.397	1.525	1.197
	192	0.509	0.4	**0.381**	0.439	0.452	1.931	1.635
	336	0.56	**0.381**	0.4	0.487	0.486	1.835	1.604
	720	0.682	0.455	**0.436**	0.474	0.511	1.625	1.54
ETTm1	96	0.389	**0.335**	0.348	0.419	0.475	0.571	0.546
	192	0.421	**0.358**	0.375	0.441	0.496	0.669	0.7
	336	0.845	**0.38**	0.388	0.459	0.537	0.871	0.832
	720	0.851	**0.414**	0.422	0.49	0.561	0.823	0.82
Electricity	96	0.477	**0.217**	0.237	0.308	0.317	0.368	0.357
	192	0.372	**0.242**	0.248	0.315	0.334	0.386	0.368
	336	0.435	0.277	**0.265**	0.329	0.338	0.394	0.38
	720	0.561	0.383	**0.297**	0.355	0.361	0.439	0.376
Exchange	96	0.513	0.225	**0.208**	0.278	0.323	0.752	0.812
	192	0.628	0.333	**0.3**	0.38	0.369	0.895	0.851
	336	0.694	0.814	**0.415**	0.5	0.524	1.036	1.081
	720	1.07	1.702	**0.78**	0.841	0.941	1.31	1.127
Traffic	96	0.493	0.298	**0.279**	0.366	0.388	0.391	0.384
	192	0.349	**0.236**	0.284	0.373	0.382	0.379	0.39
	336	**0.227**	0.261	0.29	0.383	0.337	0.42	0.408
	720	0.495	**0.264**	0.307	0.382	0.408	0.472	0.396
Weather	96	0.889	0.321	**0.232**	0.296	0.336	0.384	0.49
	192	0.956	0.394	**0.269**	0.336	0.367	0.544	0.589
	336	1.09	0.452	**0.301**	0.38	0.395	0.523	0.652
	720	0.714	0.479	**0.348**	0.428	0.428	0.741	0.675

comparison with the NLinear models. Below we will briefly comment on the results for each dataset, referring to the general table of results and the individual graphs showing the performance of the different predictions compared with actual values:

1. Figure 9. Looking at the graph, it is clear that the pattern has been broadly interpreted by the model. However, the model emphasizes some features of it by making mistakes. See how the downward movement on December 28 is also caught by the model, which, however, exaggerates it. The same downward movement is also repeated by the model the next day, but not by the actual data except to a very small extent. Evidently the model has "memory" of this downward movement and repeats it even though it almost did not actually occur. And again observe how on the next day the model, with greater emphasis when the forecast horizon is shorter, reacts by shortening the downward movement and how the actual data on that day repeat it more emphatically than the day before. Finally on the last day of the example the model and the actual data almost coincide.
2. Figure 10.Looking at the graph, it can be seen that the model interprets with good similarity the trend in reality, somewhat better than the previous dataset. This better ability is also reflected by the numerical summary values of the MAE shown in the table (Table 16). In this example, the peaks in the actual data are approximated mainly by default by the model, especially on December 30, 2017 where the downward movement is not predicted. Recalling the previous example, there is a profound difference with the December 27, 2017 peak of ETTh1 that was instead predicted, albeit to a slightly different extent from reality, by the model and repeated in the forecast the following day even though it was not actually there

Figure 9: figure

This image presents the various test results of model One Layer on dataset ETTh1 for different forecasting windows



that day. Returning to ETTh2 of Dec. 30, 2017, the fact that the model did not predict it, despite clearly having the ability to do so, probably means that this movement does not represent a repeated pattern on previous days.

3. Figure 11. Looking at the graph shows less variability than the previous data. ETTm1 in this example varies by 0.5 units compared to about 2 in the previous two graphs. This needs to be taken into account while looking at the graph because the distance between the forecast lines with that of reality weighs less than in the previous two examples since the pattern is much more compact. In any case, the visible differences between forecast and actual trend are more pronounced than in the two previous graphs, although this does not then correspond with the comparison of the various MAEs which for ETTm1, the graph commented on here, is slightly better than ETTh2 which had already been better than ETTh1. It is also observed that here there are forecasts with a very long time horizon of 1344 and 2688. It is observed from the graph that these two time horizons are far from the line of actual values. However, the mean numerical values of the MAE are not particularly high (**??**), a sign that on average, however, the forecasts perform only slightly worse than at shorter time horizons.
4. Figure 12. Observation of the graph suggests that for this dataset the model did not interpret the patterns in the actual data. The average MAE value does not let (Table 7) guess this inability, but observation of the graph leaves no doubt that the model is far from having correctly interpreted reality. Already the time horizon of 24 values deviates greatly from the actual data by making in the jumps between periods clearly visible. It had already been observed that this dataset has an auto-correlation showing the absence of periodicity, and this is consistent with the model's inability to interpret its movements not assisted by

Figure 10: figure

This image presents the various test results of model One Layer on dataset ETTh2 for different forecasting windows

clear patterns.

5. Figure 13. Looking at the graph and also at the numerical values of the MAE (Table 8 it is clear that the model is far from interpreting reality. The various graphs on all time horizons cannot even represent a trend so erratic are they. The chart patterns of the actual trend are not followed by any model lines at all.
6. Figure 14. Looking at the graph, one can guess that the different forecast distances do not differ much from each other. Looking at the numerical values (Table 11) the intuition that there is a remarkable similarity between the different graphs is confirmed. All the graphs also have considerable outliers that are greater than the actual trend, at least in this section of the data. Overall, however, there seems to be a fair amount of predictive ability.
7. Figure 15. Looking at the graph, one can guess that the prediction model was able to correctly interpret the pattern of the actual data, even with particularly long prediction distances of 2688 values. This ability, as was the case with other datasets, is not intuitable by looking at the numerical values of the mean MAE (Table 9). All the model graphs, regardless of the time prediction distance, have no particular outliers and follow the actual graph. It also happens, as is to be expected, that the shorter the time horizon, the closer in avarage the relative graph is to the actual figure.

As you can see, the results between the slightly better model proposed in this work ("One layer") and NLinear, taken as a reference, but as mentioned the other models have practically the same results, are similar and in fact comparable. A comparison with the Persistent model is also added, but only as a comparison to some models based on the Transformer. The conclusion that the results between "One Layer" and "NLinear" are similar

Figure 11: figure

This image presents the various test results of model One Layer on dataset ETTm1 for different forecasting windows

is not surprising because two simple linear models can evidently arrive at similar results. The most different results are in the ILI and Weather datasets, for no apparent reason.

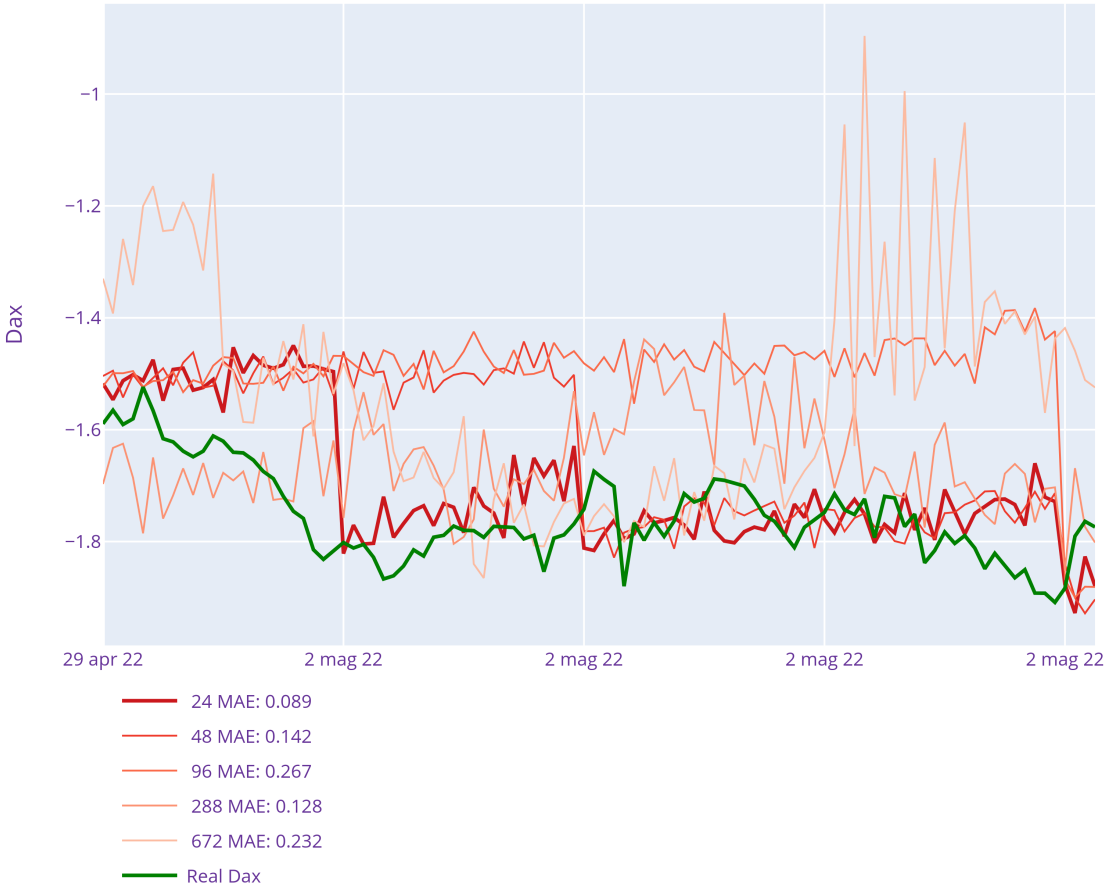
There are other issues worth addressing regarding the possible scarcity of the amount of data available in towed datasets. In [18], p. 7 the authors posed the question of whether the short length of the dataset was a serious limitation to the performance of the models. The authors' conclusion is that it is not. This is quite consistent with the observations made so far in this work provided that the towed dataset has a very regular periodicity. In fact, if the dataset has an almost unchanged periodicity for the entire dataset, it is not necessary to have large amounts of data for the models during training to fix it in the calculated weights.
A similar topic is the comparison of results obtained not on the entire dataset, but on portions of it. That is, one wants to understand whether the results on models drawn on different parts of the dataset have similar results or are different and in this case how big the differences are. To test the behavior of the results along different traits, this work considered three:

1. La Table 17 shows,for the Electricity dataset, the different results for 7 lengths of 1024 values each. The stretches are only 1024 values each.

2. La Table 18 shows, for the dataset of high water in Venice, the different results for 7 sections of 7000 values each. The hits are rather short compared to the total values of the dataset and especially compared to the training performed with all 70,000 values.

Figure 12: figure

This image presents the various test results of model One Layer on dataset Dax for different forecasting windows



3. La Table 19 shows, for the Traffic dataset, the different results for 10 lengths of 1024 values each. Traits are only 1000 values each.

Looking at the results, it is evident that the results change depending on the data segment taken into consideration and also that the best results with different forecasts belong to different segments and/or of different lengths. This implies that the results are highly dependent on the specific trait of the data set. This result is unfortunately not too surprising and is far from being an ideal result. It unfortunately demonstrates the limitations of deep learning analysis. Indeed, noticing that the same model has such different results along different parts highlights a certain degree of precariousness of the results obtained. Furthermore, there is an important consequence and that is that comparing the training results of numerous datasets in relatively large tables, comparing errors, e.g. MAE, which differ by a few units to the second or even third decimal place, for example, does not appear to make any practical sense given the great variability of the results.
In fact, not only is there some variability between different sections of the same dataset, but this variability persists even if one considers sections of different lengths even overlapping each other in the sense that they have parts in common. In this work, the MAE was calculated in different sections of increasing length. For example, the table Table 20 shows the results as the length of the section increases. Each length is specified in each column of the 'Train' row. As can be seen, there are some fluctuations that are difficult to explain from a theoretical point of view. Furthermore, it occurs that the best results all belong to the test that has the training section consisting of 13,952 values, for no apparent reason.

Figure 13: figure

This image presents the various test results of model One Layer on dataset Bitcoin for different forecasting windows

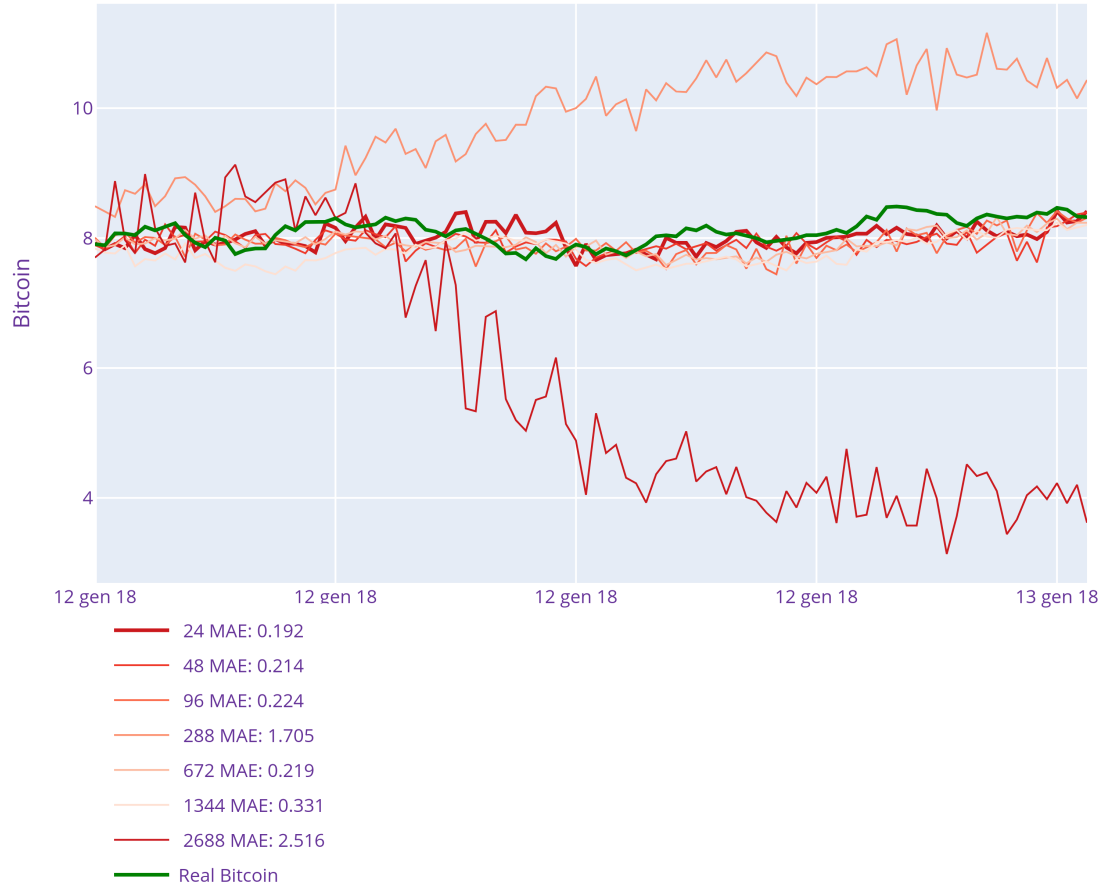


Table 17: Electricity parts MAE - Bold shows the best results for each row

Forecast	1° part	2° part	3° part	4° part	5° part	6° part	7° part
96	**0.226**	0.232	0.284	0.276	0.265	0.334	0.252
192	0.292	**0.269**	0.329	0.314	0.29	0.423	0.274
336	0.286	0.287	0.431	0.337	0.315	0.441	**0.277**

8. Conclusions

This work has set itself the goal of verifying the actual ability of certain Neural Network models to predict the trend of Historical Series even for very long-term forecasts. Starting from a very simplified Transformer-based model, even more simplified models are introduced.
The first conclusion of some relevance concerns the impact on the goodness-of-fit results of the presence or absence of the Attention modules. The conclusion is that, with the exception of a few specific datasets (bitcoin and airlines), models containing Attention modules applied to Time Series do not seem to add predictive power

Figure 14: figure

This image presents the various test results of model One Layer on dataset Temperature of Milan for different forecasting windows

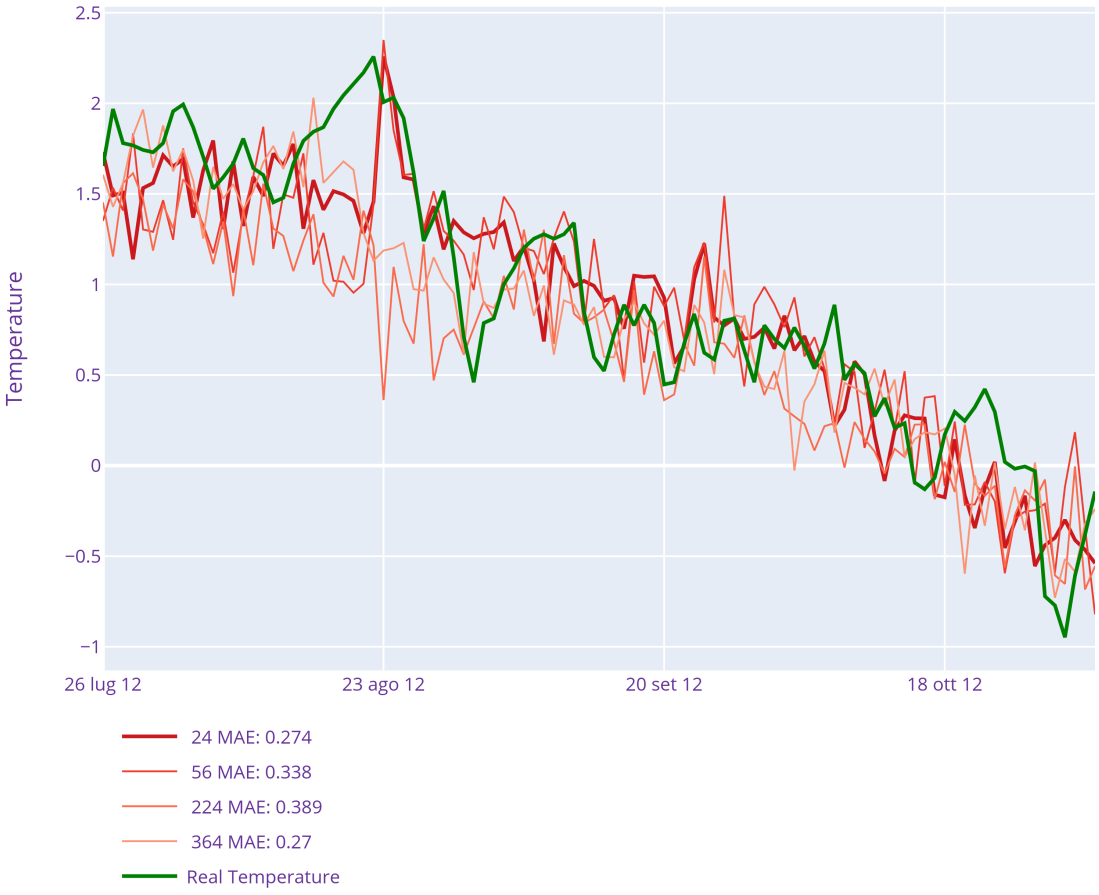


Table 18: Venezia parts MAE - Bold shows the best results for each row

Forecast	All	1° part	2° part	3° part	4° part	5° part	6° part	7° part	8° part	9° part	10° part
24	0.184	0.344	0.237	0.185	0.176	0.232	0.274	0.258	0.195	**0.165**	0.258
48	0.222	0.402	0.285	0.218	0.203	0.276	0.306	0.325	0.235	**0.188**	0.302
96	0.262	0.416	0.326	**0.236**	0.238	0.351	0.334	0.314	0.247	0.251	0.361
288	0.309	0.492	0.394	0.349	0.254	0.337	0.341	0.544	0.358	**0.212**	0.347
672	0.337	0.478	0.422	0.383	0.258	0.326	0.351	0.738	0.381	**0.223**	0.349
1344	0.339	0.445	0.562	0.43	0.34	0.321	0.349	0.764	0.516	0.439	**0.285**
2688	0.346	0.404	0.519	0.482	0.403	0.374	**0.323**	0.719	0.553	0.564	0.338
5376	0.351	0.461	0.526	0.419	0.432	0.46	**0.337**	0.684	0.506	0.48	0.449

and their presence. does not bring benefits. However, it is also predictably observed that the model is able to adapt to their presence making the Attention module almost, but not quite, irrelevant.
Several models have been studied that starting from a Transformer model progressively eliminated parts and ended up defining an extremely simple linear model consisting of a single layer.
The rather unexpected conclusion is that all these models have more or less the same predictive capacity. The differences between the results obtained are not so significant. However, a slightly better predictive ability, on average, can be noted in the simpler models.

Figure 15: figure

This image presents the various test results of model One Layer on dataset Venezia See Level for different forecasting windows

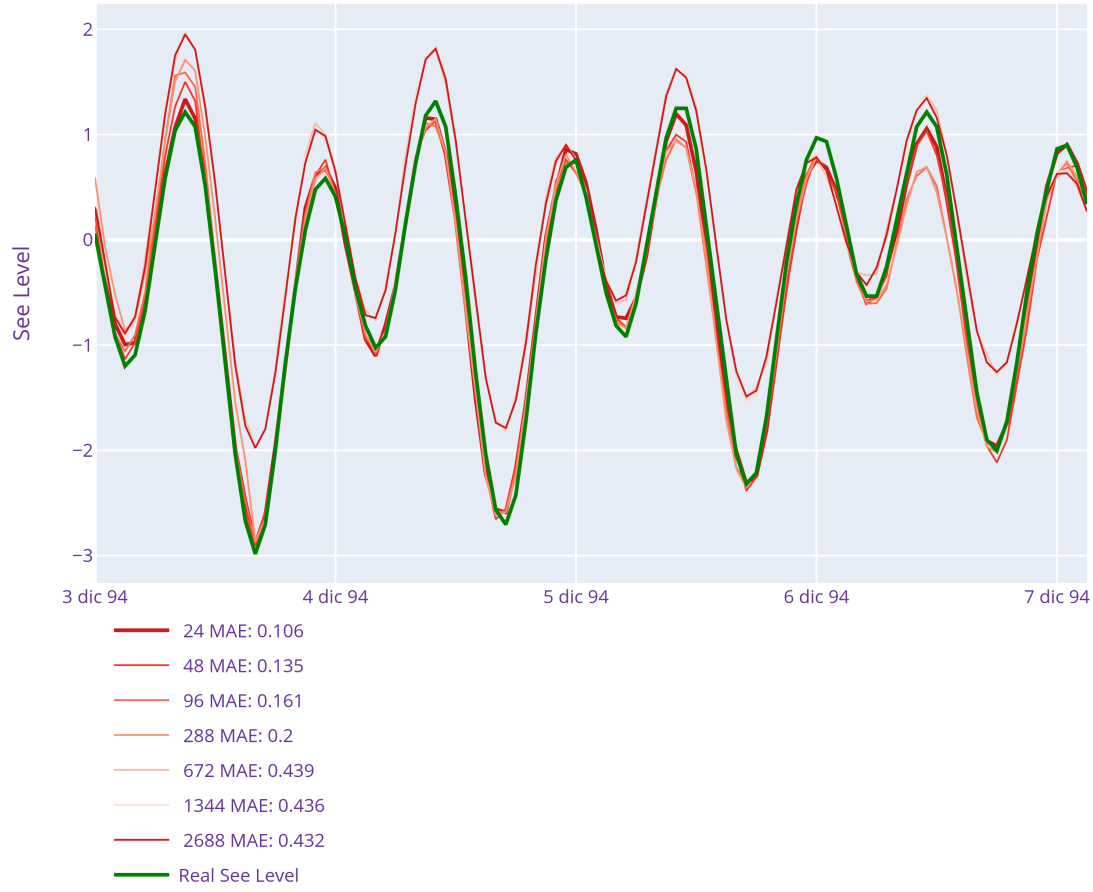


Table 19: Traffic parts MAE - Bold shows the best results for each row

Forecast	All	1° part	2° part	3° part	4° part	5° part	6° part	7° part	8° part	9° part	10° part
96	0.333	0.379	0.298	0.324	0.299	0.303	0.35	0.349	0.309	0	0
192	0.253	0.301	0.236	0.28	0.281	0.259	0.275	0.35	0.272	0	0
336	0.273	0.384	0.263	0.261	0.263	0.28	0.281	0.288	0.428	0	0
720	0.28	0.383	0.275	0.264	0.264	0.307	0.304	0.331	0.367	0	0

Furthermore, it has been shown that for data sets with regular periodicity, the simple model consisting of a single linear layer retains the predictive capability even for very long periods, and we have demonstrated this for long periods of up to 5376 values. The conclusion is that simple neural networks are able to repeat periodic patterns and are able to anticipate the future, even with the limitations we have tried to highlight.

Table 20: Venezia growing parts MAE - Bold shows the best results for each row

Forecast	All	1° part	2° part	3° part	4° part	5° part	6° part	7° part	8° part	9° part	10° part
96	0.262	0.335	**0.223**	0.261	0.274	0.264	0.294	0.3	0.277	0.275	0.269
192	0.293	0.35	**0.242**	0.293	0.302	0.308	0.357	0.352	0.326	0.316	0.303
336	0.317	0.373	**0.252**	0.313	0.316	0.335	0.4	0.393	0.368	0.347	0.325
720	0.338	0.385	**0.262**	0.334	0.335	0.368	0.438	0.438	0.423	0.38	0.345
Train	69888.0	**6912.0**	13952.0	20992.0	28032.0	34944.0	41984.0	49024.0	56064.0	62976.0	70016.0

Acknowledgements

I would like to thank Prof. Matteo Matteucci who gave me confidence by assigning me a very interesting thesis, assisting me with his valuable advice, which allowed me to conclude my engineering studies at the Politecnico di Milano in such a pleasant way. A special and very big thank you goes to Eng. Eugenio Lomurno who patiently followed me in the various elaborations of this paper with constant presence combined with many, precious and very welcome advices. Finally, a collective thank you to all the professors and tutors I have encountered throughout my studies. I will forever have a most pleasant memory of them as extraordinary people who do their work with passion and extraordinary ability.

Ringraziamenti in lingua italiana

Desidero ringraziare il Prof. Matteo Matteucci che mi ha dato fiducia assegnandomi una tesi interessantissima, assistendomi con i suoi preziosi consigli, che mi ha permesso di concludere in maniera così piacevole gli studi di ingegnaria al Politecnico di Milano.
Uno speciale e grandissimo ringraziamento va all'Ing. Eugenio Lomurno che con pazienza mi ha seguito nelle varie elaborazioni di questo scritto con costante presenza unita a tanti, preziosi e graditissimi consigli.
Infine un ringraziamento collettivo a tutti i professori e a tutti gli esercitatori che ho incontrato durante l'intero ciclo di studi. Ne avrò per sempre un ricordo piacevolissimo di persone straordinarie che svolgono il loro lavoro con passione e capacità straordinarie.

9. Bibliography and citations

References

[1] Fatoumata Dama and Christine Sinoquet. Time series analysis and modeling to forecast: a survey. arXiv:2104.00164v2, 2021.

[2] Timo Denk. Linear relationships in the transformer's positional encoding. *Annalen der Physik*, 2019.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 2015.

[4] Justin Johnson. Deep learning for computer vision. 2019.

[5] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, and Pascal Poupart andMarcus Brubaker. Time2vec: Learning a vector representation of time. 2019.

[6] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer : The efficient transformer. 2020.

[7] Ivano Lauriola, Alberto Lavelli, and Fabio Aiolli. *Computing Reviews*, 24(11):503–512, 2021.

[8] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. 2019.

[9] Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X. Liu, and Schahram Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. 2022.

[10] Huanru Henry Mao. A survey on self-supervised pre-training for sequential transfer learning in neural networks. 2020.

[11] Maithra Raghu and Eric Schmidt. A survey of deep learning for scientific discovery. 2020.

[12] Christoph Sager, Christian Janiesch, and Patrick Zschech. A survey of image labelling for computer vision applications. *Journal of Business Analytics*, arXiv.2104.08885, 2021.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. arXiv:1706.03762, 2017.

[14] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, and Junchi Yan Liang Sun. Transformers in time series: A survey. 2022.

[15] Wikipedia. Time series - wikipedia. 2022.

[16] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. 2021.

[17] Neo Wu, Bradley Green, Xue Ben, and Shawn O'Banion. Deep transformer models for time series forecasting: The influenza prevalence case. 2020.

[18] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? *arXiv preprint arXiv:2205.13504*, 2022.

[19] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. 2021.

[20] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. 2021.

[21] zhouhaoyi. Etdataset, 2020.