POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

A Transfer-learning enabled Transformer for time-series prediction

Laurea Magistrale in Computer Science - Ingegneria Informatica

Author: Diego Riva

Advisor: Prof. Manuel Roveri

Co-advisor: Alessandro Falcetta

Academic year: 2020-2021

1. Introduction

Thanks to the availability of large collections of data and an increasing interest on the task, the Time Series forecasting problem became a key aspect inside a lot of realities like in economics, finance, production, advertising, and social policies.
Deep Neural Networks (DNNs) are some of the most recently introduced techniques in the Machine Learning field. These models brought great results in time series analysis, performing particularly well when a large enough amount of data is available.
Among these, te last innovation is an attention based model: the Tranformer. It has been developed to perform Natural Language Processing (NLP) tasks, in which, it represents the current state of the art.
The first objective of this work is to implement a Transformer-based model able to effectively increase the performance in the Time Series prediction task in different domains.
Moreover, still there are a lot of cases where training data is expensive or difficult to collect. Therefore, we tested how a transfer learning approach can improve the results when the target dataset has a low number of samples but we have at our disposal a larger correlated dataset on which perform the training, and then, a fine-tuning of the model.

2. Background

A collection of data ordered by timestamp is defined as time series: for example the temperature of the previous days, the COVID-19 spreading or economic indicators. We are going to train a model able to find patterns that allows to predict the next value of a time series. It is, given $X = [X_1,...,X_n]$, an ordered set of real values, we want to obtain the best possible predictor for X_{n+1}, i.e. $\hat{X}_{n+1}$.
Historically, this task has been performed by statistical models, among which one of the most common is the *Autoregressive Integrated Moving Average* (ARIMA) model. The three modules of which it is composed (Autoregression, Integration, Moving Average) allow to handle non-stationary time series, thanks to the differencing step, and then to interpret samples based on their relationship between the past observation and their moving average. During the 90's with the spreading of the Artificial Intelligence (AI) and the coming of Deep Learning (DL), *Recurrent Neural Networks* (RNNs) [11] have been an important focus of research and development for processing sequence of values.

In RNNs each output is the result of a function applied on the previous output with the same rule, such that we can create a cyclic graph in which the actual value of a variable influence the future value of the same variable. It generalizes the auto-regressive concept of ARIMA models, using distributed hidden layers that store information with non-linear dynamics. However, to train an RNN on long sequences, we must run it over many time steps making the RNN a very deep network resulting in vanishing gradient problems. To mitigate this drawback, *Long Short Term Memories* (LSTM) [8] introduce self-loops to produce paths where the gradient can flow. More specifically, each cell is composed by four main layers:

- a tanh activation that receives the input and the previous state.
- forget gate: controls which part of the previous long-term state can be erased.
- input gate: controls which part of the main layer output should be added to the long-term state.
- output gate: controls which part of the long-term state should be read and output at the current time step.

This model has been so successful in a multitude of applications like power consumption [9], trajectory prediction [1], traffic forecast [15],financial time series forecasting [5] or in medical fields [6]. However it still has an hard time learning long-term patterns in which a data information have to be carried over many steps. The last innovation in DL field is the *attention mechanism* introduced in [3]: it focuses on the appropriate part of the time series at each time step, thus reducing the path from an input to its dependencies, such that the short term memory limitations of RNNs have much less impact. Initially applied in neural machine translation (NMT), this mechanism become a fundamental operation in a sequence to sequence architecture, allowing the decoder module to receive not only the last encoder hidden state, but also all of its inputs. This will make the decoder determine which part of the time series to focus each time step.

In 2017, Vaswani et al. [12] proposed a sequence-to-sequence architecture called *Transformer* which improved the state of the art in NMT without using any recurrent or convolutional layers, just the attention mechanism. The main advantages are that this architecture is faster to train and easy to parallelize, but introducing a space complexity problem with respect to the input size. Since its release, initially designed to handle NLP problems, the Transformer model has been subject of a series of further studies that increased performances and explored applications on new tasks and fields: in [10] the authors proposed a convolutional self-attention to further improve the accuracy and a sparse attention mechanism to reduce the space complexity; recently [4] a Longformer model has been introduced to address the problem of the space complexity: it scales linearly with the sequence length. It is a drop-in replacement for the standard self-attention and combines a local windowed attention with a task motivated global attention obtaining state of the art results on document tasks; [14] applied the Transformer model as case-study in a time series prediction task underlying how this model can be a promising alternative to the previous state of the art reached by LSTM.

2.1. Transfer learning

The idea is to store the knowledge about a specific task in a model that will be reused for another task, different but similar to the previous one. The main advantage is that in this way it is possible to perform tasks on small datasets by exploiting the inner dependence between two similar dataset. This kind of approach had a great success in the image classification and in the NLP tasks: in [7], the authors demonstrate the effectiveness of self-supervised pretraining on a large corpus, using a transformer-like architecture. Furthermore recent study of Sandra Wankmuller [13] analysed transformer-based models for transfer learning in the classification task. However, the majority of studies underline how important is the correct choice of the training dataset: choose a dataset that is not correlated enought with our target will likely bring to worse results with respect to training directly on the target one, since the model would not be able to correctly interpret patterns and use the knowledge acquired by the training dataset. Our aim is to proceed in this direction, evaluating how a transformer model can benefit from transfer learning in a time series prediction task.

3. The model

Our transformer model is based on the original transformer proposed by [12] composed by an embedding, an encoder and a decoder layers (Fig.1).

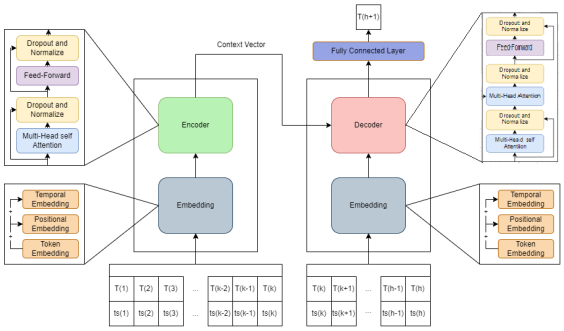

Figure 1: Proposed Transformer-based model
Representation of the proposed model with input and output sequences.

3.1. Embedding

Initially, the raw input sequence needs to be transformed in a vector with size equal to the model's dimension. Our aim is not only to consider the positional dependencies, but also to be able to catch temporal dependencies that varies from a specific application case to another. To achieve this behaviour, we have defined the Embedding Layer as the sum of three main components:

- Token Embedding: it is a 1D-convolution (or *temporal convolution*) that takes the input vector over a single temporal dimension to produce a tensor in output with size equal to the dimension of the model.
- Positional Embedding: it is a Dense layer that encodes the position of a word in a sentence in a unique value, such that the model is able to take into consideration the order of the feature's values.
- Temporal Embedding: it encodes the timestamp value of the input couple feature-timestamp, and returns a vector of float values that can be used to help the model to catch temporal patterns. Our implementation allows two different configurations: the *Fixed* configuration, that return a dense vector of fixed and unique values for each value in the input sequence; it is dependent by the position, so the same position in input will have the same codification. And the *Dynamic* configuration in which the dense vector is no more dependent by the input position but only by the date value.

3.2. Encoder

The Encoder layer is the component of our model that has to understand the context of the current input. In our work it means that it should figure out the environment, from which the last observation has been generated, looking at the past values. As in the literature it is composed of 4 modules stacked as follow:

- multi-head self attention module: this is the most relevant component on which the model is based on. One of the weakness of the canonical attention mechanism is its limit to catch local context, that makes the model more vulnerable to anomalies [10]. So, in this work, we compare the performances in terms of loss between the original matrix multiplication, with a convolution of kernel size k and stride 1.
  The input pass through each head in parallel, to output a series of attention matrices that are first concatenated and then linearly transformed by a fully connected layer.
- dropout and normalization layers.
- feed forward layer: it is a functional layer whose weights are learned during training and the same matrix is applied to each token's position. It processes the output of the previous module to better fit the next attention layer.
- dropout and normalization layers.

By construction, it is possible to stuck N encoder layers in series (in our setting we choose $N = 1$).

3.3. Decoder

The Decoder receives both the Encoder output and the portion of the input sequence that contains the last observation. The Decoder is composed by a stack of the layers already described in the previous subsection. In the implementation we adopted the same strategies and decisions analyzed before. After the embedding transformation, the Decoder input sequence is passed into the first multi-head attention mod-

ule. After the canonical dropout and normalization layers, there is another Multi-Head Attention layer that receives the outputs of the Encoder and the previous Attention that are mapped together to find out the relation between the context and the current observation.
Finally the resulting tensor is fed into the last fully connected layer that produces the output of the model that will be post-processed to return in the source domain.

4. Data and training

We used three different dataset to catch the performances of the model in different kind of domains:

- Covid-19 cases: it is a dataset of small size that registers the number of new daily cases.
- Stock exchanges: it is a dataset with time series that involves some major stocks that come in the S&P500 ETF. In our study we will consider the "closing daily price" as feature to analyze.

The datasets are initially differenced with a lag of s steps, where s changes based on the case study. Then the from the resulting sequence we construct a tensor of shape (*batch_size, input_length, feature_size*). Then the model's input is a window of that tensor with length equal to the *input_length* hyperparameter and is composed by couples (T_t, ts_t): T_t is the feature's value at time t, while ts_t is the timestamp corresponding to the observation at time t.
Since this choice would involve to be in a multivariate problem, we set a first embedding phase that produces a codified float value representing the timestamp. This one will be then summed up with the other feature to incorporate the timestamp inside the feature value as a one-dimensional problem.
During learning the original dataset is divided in two consecutive subsets A and B. Since we are going to compare our prediction metrics with those of an our previous work [2], we set the learning and prediction phases works as follow:

1. initialize the model
2. train on A.
3. the trained model predict the next time step.
4. the prediction is evaluated against the known value from B.
5. the known value is then included in A, updating mean and standard deviation on A to adjust the standardization.
6. the process is repeated until A coincides with the original dataset.

In a typical training setup we want to predict the next value, considering the 28 previous steps as in (fig 2): given the encoder input $[X_1...X_{22}]$ and decoder input $[X_{22}...X_{28}]$ the model will predict X_{29}.

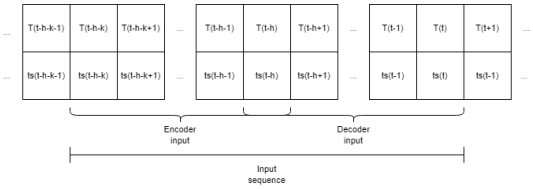

Figure 2: Input window
In the figure the input of our model: the sequence length is a window of size *k+h-1* that flows over the time series step by step. We have that k is the length of the Encoder input while h is the length of the Decoder one.

Lastly, our aim is also to test how the model performs in a transfer learning scenario. What we do is to perform a Dynamic Time Warping algorithm over multiple datasets to calculate the distance measure between them. It produces a similarity metric able to minimize the effects of shifting and distortion in time by allowing "elastic" transformation of time series in order to detect similar shapes with different phases.
This kind of approach allow us to test how the difference between two time series affect the performances of transfer learning following the considerations made in the paragraph 2.1.
In this work, then, we perform and compare three different scenarios:

- train the model on the original dataset.
- train the model on the original dataset and then use this model to predict another time series.
- train the model on the original dataset and then fine tuning only the Decoder module over another time series.

5. Results

In this section we present the performances of our algorithm in the various configurations that we have mentioned in the previous sections. First of all in Table.1 we report Mean Absolute Error (MAE) of our transformer's predic-

	COVID-19 ITA	AAPL
range	7224	43.65 - 143.16
TS length	495	1250
transformer	364.12	0.7266
TIMEX-prophet	489.05	2.5586
TIMEX-exp.smooth	380.33	1.3221
Persistence model	/	0.7749

Table 1: Prediction results
On the rows the models used for the comparison over the three chosen dataset: the Persistence Model has been tested only in the stock dataset due to its relevance in this case study. To measure our accuracy we calculate a Mean Absolute Error (MAE) over the prediction and the ground truth.

	COVID-19: ITA			
	BRA		GBR	
range	115128		54183	
length rate	30	50	30	50
TL no fine tuning	20344.2	18939.0	2985.6	2230.8
TL with fine tuning	8662.4	5595.9	2152.9	2346.4
no TL	7154.4	3905.5	2625.7	2051.6

Table 2: Transfer Learning COVID-19
In the table the results of the transformer model over a portion of the target dataset which length is expressed in 1. The values in cells are the MAE in the three different analyzed situations.

tions over the two datasets, in comparison with the performance of the Prophet and Exponential Smoothing models analyzed in an our previous study [2].
The model produces performances similar to our benchmark on the COVID-19 dataset while it outperforms the other models on the AAPL time series. This behaviour id probably due to the restrict number of sample in the COVID-19 dataset that limits the performance of a complex model.
Then we tested how the Transformer can exploit the transfer learning task, which results follows in Table.2. The DTW test returns that the GBR and AMZN are the most similar target datasets to the ITA and AAPL sources respectively, while BRA and ORCL are the most different. This aspect is fundamental to interpret the results since in both cases the results of transfer learning with fine tuning is the best performer when size of the target dataset and the DTW distance are very small. The opposite happens when the source and target datasets are dissimilar or the length of the last one is big enough to allow the ad-hoc model to reach better results.
Finally, to validate our proposed Transformer, we tested two main variants:

- Convolutional attention mechanism: instead to use the typical Dense layer to represent the "value", "query" and "key", we use a one dimensional convolution.
- Remove the Temporal Embedding Layer: one of the major components of the model is the embedding layer that prepares the input to be given to the Encoder. It's rule is to include the timestamp information inside the sequence, evidencing time patterns that could represent time dependencies.

Results follows in Table.4 and points out how the two variants behave in a different way in the two analyzed datasets but maintaining stable the overall performances. We can't prove their effectiveness in a general manner but their use is dependant by the case study and the in-

	stock: AAPL			
	AMZN		ORCL	
range	1189.2 - 3471.3		46.63 - 80.270	
length rate	30	50	30	50
TL no fine tuning	64.187	58.674	2.1395	2.1277
TL with fine tuning	29.776	27.331	0.7746	0.8032
no TL	41.330	30.543	0.6743	0.5409

Table 3: Transfer learning stocks
In the table the results of the transformer model over a portion of the target dataset which length is expressed in 1. The values in cells are the MAE in the three different analyzed situations.

	COVID-19: ITA	stock: AAPL
Original Transformer	364.12	0.7266
Convolutional	342.76	0.8364
No-Temporal embedding	347.53	0.7944

Table 4: Model variants MAE

trinsic characteristic of the time series.

6. Conclusions

In this thesis we presented a Transformer model adapted to handle the time series prediction task. We tested its performances in comparison with a benchmark obtaining promising results, mostly when we used a large dataset.
Then, we tested how this kind of architecture is able to generalize the problem enough to be employed in a transfer learning approach. Also in this application, the model seems to provide interesting results, especially when the source dataset is large enough to allow a solid generalization, and when there is a strong similarity between source and target dataset. Finally we examined some variants of the proposed model: the use of convolutional attention, instead of the original fully connected attention, seems to work better when we want to penalize outliers but it risks to introduce an error when the dataset is very fluctuating. We also analyzed the effectiveness of our Temporal Encoding that embeds the timestamp (when available) in the sequence: in this case, the dataset that present a shape that does not highlight any particular time patterns seems to benefit more from this strategy, with respect to a dataset that may already have strong time patterns that is not necessary to accentuate more.
In conclusion we have proved that the Transformer, originally applied to NLP task, is a model that can find its application also in the time series prediction task providing competitive performances in various domains and when adopted in strategies like transfer learning. In this prospect it would be interesting to test the proposed model on larger datasets and in other case studies to validate this work. Moreover, a limit was represented by the chosen training strategy that optimizes the performances but introduces an heavy computational cost: it could be interesting to analyze how a training window over the dataset, or strategies that avoid the continuous retrain of the model, can impact the time cost and the performances. Finally, more tests need to be made for the temporal embedding, to understand when it is beneficial and if it could be applied in time series that have missing values to attenuate the lack of information.

7. Acknowledgements

Here you might want to acknowledge someone.

References

[1] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm:

Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 961–971, 2016.

[2] Falcetta Alessandro and Roveri Manuel. TIMEX. `https://github.com/AlexMV12/TIMEX`, 2021.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[5] Jian Cao, Zhi Li, and Jian Li. Financial time series forecasting model based on ceemdan and lstm. *Physica A: Statistical Mechanics and its Applications*, 519:127–139, 2019.

[6] Vinay Kumar Reddy Chimmula and Lei Zhang. Time series forecasting of covid-19 transmission in canada using lstm networks. *Chaos, Solitons & Fractals*, 135:109864, 2020.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] Weicong Kong, Zhao Yang Dong, Youwei Jia, David J Hill, Yan Xu, and Yuan Zhang. Short-term residential load forecasting based on lstm recurrent neural network. *IEEE Transactions on Smart Grid*, 10(1):841–851, 2017.

[10] 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. *Advances in Neural Information Processing Systems*, 32:5243–5253, 2019.

[11] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[13] Sandra Wankmüller. Neural transfer learning with transformers for social science text analysis. *arXiv preprint arXiv:2102.02111*, 2021.

[14] Neo Wu, Bradley Green, Xue Ben, and Shawn O'Banion. Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv preprint arXiv:2001.08317*, 2020.

[15] Zheng Zhao, Weihai Chen, Xingming Wu, Peter CY Chen, and Jingmeng Liu. Lstm network: a deep learning approach for short-term traffic forecast. *IET Intelligent Transport Systems*, 11(2):68–75, 2017.