University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Marta Nowakowska
Student no. 385914

Time series prediction using self-attention models

Master's thesis
in COMPUTER SCIENCE

Supervisor:
dr hab. Marek Cygan, prof. UW
University of Warsaw

Co-supervisor:
dr hab. Piotr Miłoś
Institute of Mathematics, Polish Academy of Sciences

Warsaw, August 2022

Abstract

TODO abstract

Keywords

time series, machine learning, self-attention model

Thesis domain (Socrates-Erasmus subject area codes)

11.4 Sztuczna inteligencja

Subject classification

D. Software
D.127. Blabalgorithms
D.127.6. Numerical blabalysis

Tytuł pracy w języku polskim

Prognozowanie szeregów czasowych przy użyciu modeli z uwagą

Contents

Chapter 1

Introduction

1.1. Time series and their importance

Forecasting future values and their probability distribution on the basis of historical datapoints is one of the most prevalent problems in both industry and research. For example, time series analysis can be used for predicting the number of ride-hailing trips completed during special events like holidays ([10]), but also for estimating stock market behavior ([11]) and solving resource allocation tasks ([12]).

This wide usage brought time series a lot of attention in the past. One of the most known methods for their forecasting are based on Box-Jenkins method like the ARIMA (autoregressive integrated moving average) [8] model, but there also exist ones using the exponential smoothing techniques and state space models [9]. Despite their popularity, they come with some fundamental disadvantages. For example, they need to be fit separately for every similar sequence ([1], [7]) and cannot be trained to recognize these similarities which makes using them time-inefficient in the long term. Furthermore, these models typically use parameters that have to be handpicked by the user ([1], [7]), which makes them more suitable for field experts with knowledge of both parameter selection procedure and preferably time series' behavior.

The aforementioned issues are not a problem for deep neural networks, since they can be trained to recognize a group of similar time series and their hyper-parameters have to be only selected once for the training and not at the prediction time. What is more, once trained neural network can be adapted to new data using transfer learning.

On top of that, neural networks proved to be very successful in other fields, including natural language translation, which is a problem with similar properties to the time series forecasting. In particular, the translation can be defined as a mapping of one sequence to a second one, similarly to how one can map historical values to the future ones. Thus, the achievements of natural language processing (NLP) field, including RNN models have almost directly transferred to the time series forecasting. However, RNNs and their successors (LSTM [2], GRU [3]) have difficulty with long sequences [4], causing them to struggle with time series that can contain long-term dependencies.

The next important step in the development of deep neural network adaptations for the time series forecasting was the Transformer [5]. Instead of using recurrent networks it relies almost entirely on the self-attention mechanism, which can be trained to estimate dependencies in arbitrarily long sequences, making it possible for the model to deal with larger-scale forecasting. However, there are several problems with integrating it directly for the time series forecasting usage. One of them is that the scaled dot-product attention can only capture

dependencies between particular data points, but it cannot learn the time series' shape, which makes it struggle with capturing its seasonality and other patterns or anomalies ([1]). The second important issue is that its space complexity grows quadratically with the sequence length, making it difficult to train larger Transformer models on long time series ([1]). One solution to these problems was proposed in [1] by introducing convolutional self-attention and LogSparse Transformer.

1.2. Problem statement

Transformer proposed in [1] is centered around the idea of every point in a time series being drawn from some Gaussian distribution. Thus, the model's goal is to predict the parameters of these distributions, thus ensuring that each prediction is partially random and introducing Gaussian noise to it. [pk: formally?] [mn: I wanted to leave the formal definitions for the background, is that ok?]

In reality, as we show later in this work, not every time series' noise comes from a Gaussian distribution. [pk: examples? preferably with pictures] [mn: I'd probably leave it for later sections before our results or get back here if I find it's more appropriate here] It would be more useful if we could model any continuous distribution instead. One way to approximate it is by using a discrete distribution with categories that are granular enough. Obviously, this comes with a disadvantage of the model being able to predict only a discretely limited value, which could cost some accuracy. However, the expectation is that this problem can be solved easily by just using a sufficiently large number of categories.

The goal of our project was using the Transformer model to predict time series assuming that each point in the sequences belongs to a categorical distribution.

1.3. Contributions

Work described in this thesis has been done via a collaboration with scientists from Institute of Mathematics of Polish Academy of Sciences.

My most important contributions are:

1. Gaussian distribution-based Transformer
2. Experiments with the Gaussian distribution-based Transformer
3. Base implementation of the dataset feeder
4. Time features as in [1] (on the data feeding side)

Chapter 2

Background

2.1. Time series and notation

We define time series as sequence of real numbers measured across time with a constant **frequency**. Frequency is defined in time units, e.g. weeks, days, hours. For multiple time series represented as a matrix y, where each row is a separate time series, $y_t^{(i)}$ denotes t time step of a series number i.

One can extend the definition above to a sequence of any data type, but it is not necessary for the datasets that we use for this thesis.

2.2. Transformer model architecture

Transformer [5] is a deep neural network model that relies on a scaled dot-product attention mechanism introduced in the same paper. There exist some modifications to the model presented in [5], but we follow the "decoder-only" [6] architecture type.

2.2.1. Overview

Our implementation of the Transformer model consists mainly of a causal convolution layer, positional encoding layer, N identical decoder layers, layer normalization and a fully-connected layer. Most of the layers' outputs have d_{model} dimension. I present the general architecture on Figure 2.1.

Positional encoding allows us to feed the model embedded information about how the data points are ordered in the inputted time series. We use a trainable embeddings for the positional encodings and add it to the input before it reaches the first decoder layer. [pk: ???] [mn: I changed it a little, but I am not sure whether it is still confusing.]

A single **decoder** layer consists of a causal attention sub-layer and a feed-forward sub-layer:

- The feed-forward sub-layer contains a simple feed-forward neural network with two dense layers and a ReLU activation in-between.
- The causal attention sub-layer contains the attention mechanism that I describe later on.

Both sub-layers are residual and wrapped with a layer normalization and a dropout layer.

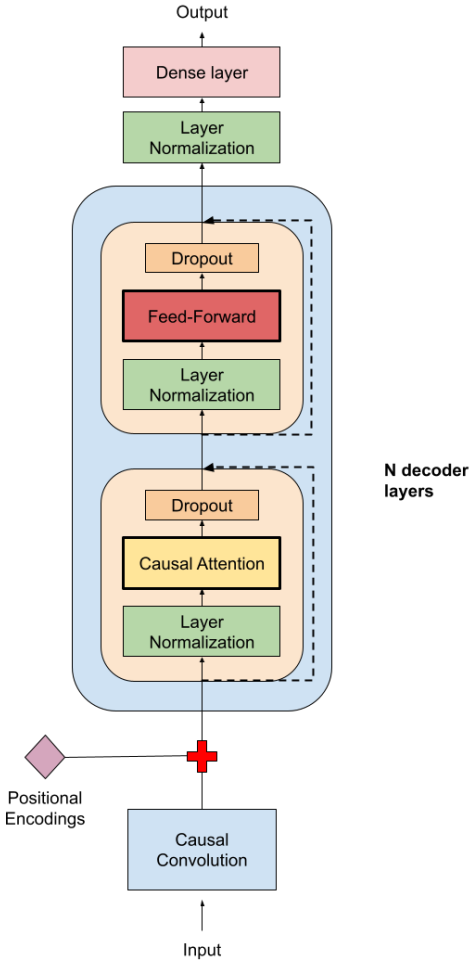

Figure 2.1: Transformer's architecture. The dotted lines represent residual connections.

2.3. Attention

2.3.1. Scaled Dot-Product Attention

Attention is the key part to explaining how the Transformer works. It enables us to train the neural network to encode the relations between different parts of the input (e.g., a time series' data points).

Scaled dot-product attention introduced along with the Transformer architecture in [5] is perhaps the most known kind of attention function. To calculate the attention score vector between one data point and all that come after, one first needs to provide a query vector for the first data point and a set of vector pairs consisting of keys and values for all the other data points. We match our query against all of the keys by a dot product and use the softmax function to normalize the result. Finally, by multiplying the softmax results with the corresponding values we get a set of weighted values vectors. After summing them we get the attention score vector for one data point.

One can informally see these calculations as operations on a neural network-approximated dictionary, where each key corresponds to a certain value and we treat the dot operation between the query and keys as a measure of similarity. When they are orthogonal, the dot product will be equal to 0. On the other hand, for vectors of fixed length the dot product

is maximal when they have the same direction. When the result of softmax is large between two data points, we can informally say that the attention relates these two points.

In reality we do not need to calculate the attention sequentially for every data point, but rather use matrix operations. One needs to provide three matrices to the attention layer: $Q, K \in \mathbb{R}^{d_I \times d_k}, V \in \mathbb{R}^{d_I \times d_v}$ (respectively called queries, keys and values). They are created by multiplying the input matrix $I \in \mathbb{R}^{d_I \times d_{model}}$ with three distinct trainable weight matrices $W_Q \in \mathbb{R}^{d_{model} \times d_k}, W_K \in \mathbb{R}^{d_{model} \times d_k}, W_V \in \mathbb{R}^{d_{model} \times d_v}$.

For given Q, K, V we compute the attention with the following formula:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}} \cdot M)V \text{ (from [1])}$$

[pk: a picture is worth a thousand words] [mn: I think here maybe it's worth adding some words, since I did not talk a lot about the informal understanding of attention. Above I added some paragraphs about how to calculate and understand it. If I were to add a picture, maybe the one from the paper (Figure 2 in Attention...)? But I think this formula is very short anyways.]

where d_k stands for the dimension of Q and K. Dividing by its square root is supposed to balance out the growth of Q and K^T dot product that occurs with larger values of d_k.

M is a **mask matrix** that sets all the elements corresponding to connections between future and past (upper triangular) to $-\infty$. It prevents the model from attempting to relate the past events to the future ones. [pk: show the matrix] [mn: I fixed it right now, since I see I was wrong before of how it should look like. Not sure whether you'd still want me to show it, since it should be one of the simplest matrices spoken about here. WDYT?]

While calculating the softmax score for any two data points in a time series, the scaled dot-product attention does not use any information about other data points. We call this property **context-blindness**. Sometimes this might lead the model to assigning strong attention score between events that have similar values, even if their contexts (neighboring data points) are very different. That might cause the model to incorrectly recognize patterns in the time series ([1]).

2.3.2. Convolutional Attention

Definition 2.3.1 ***Causal convolution*** *is a convolution used on input padded with $k-1$ zeros at the start, where k denotes the size of the kernel.*

This type of convolution is particularly useful for time series, since it enables us to make predictions with only partial input, essentially generating a completely new time series.

The convolutional attention [1] is calculated in almost the same way as the scaled dot-product attention. It only differs in the way it computes queries and keys. Instead of applying a matrix multiplication with weight matrices, it uses a causal convolution with a stride of 1. When the kernel's size is equal to 1 it is equal to the matrix multiplication.

The usage of convolution is supposed to offset the dot-product attention's context-blindness. Since the convolution layer learns to recognize input's context, stronger attention score should be assigned to similar events.

2.3.3. Multi-headed attention

Each attention block might contain more than one parallel attention layer, aka attention "head". In theory this allows for different layers to learn various kinds of relationships in the

input and is a very successful tool e.g. in natural language processing [5].

The outputs of all h attention heads are concatenated and multiplied by a weight matrix $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ to squeeze them back into $d_I \times d_{model}$ shape before passing them to the next layer.

To offset the additional computational cost related with multiple attention heads, we set d_k and d_v to d_{model}/h.

2.4. Training

2.4.1. Additional input

[pk: what is a covariate?] [mn: I changed the name since it's not really necessary to call these covariates.]

Following [1], we use additional variables as part of the model's input aside the time series itself. There are three kinds of additional input we use:

1. Numerical information about time the input was recorded (year, month, day-of-the-week, hour-of-the-day, minute-of-the-hour).
2. Identifier of the time series.
3. Input's distance from the sequence's start.

We embed all of the additional variables to the same dimension, sum those embeddings and normalize them. Finally, we add the result to our model's (embedded) input and normalize it as well.

2.4.2. Weighted sampling

We adapt a weighted sampling technique in our training, similarly to [1] and [7]. Its main purpose is to counteract model's tendency to underfit data of a large scale, since it occurs more rarely in the datasets. We test that idea by including the weighted versus uniform sampling in our grid searches.

We calculate the weights by normalizing the scale factor v_i:

$$v_i = 1 + \frac{\sum_{t=1}^{T} y_t^{(i)}}{T},$$

where T is $y^{(i)}$'s length.

2.5. Metrics

2.5.1. Quantile loss

Quantile loss is a metric designed to evaluate and/or train models designed with quantile prediction in mind. We define ρ-quantile loss as follows:

$$QL_\rho(\hat{y}, y) = max(\rho(y - \hat{y}), (\rho - 1)(y - \hat{y})),$$

[pk: picture maybe?] [mn: I would do something like here: https://towardsdatascience.com/quantile-regression-from-linear-models-to-trees-to-deep-learning-af3738b527c3 or (best) borrow it, but I am not sure how it works with licenses and copyrights?]

where $\hat{y}$ denotes the model's prediction for the ρ quantile and y is the ground truth value.

This metric is characterized by a stronger penalization of over-predictions for ρ smaller than 0.5, and under-predictions for ρ greater than 0.5. [pk: good intuition]

2.5.2. Risk metric

$Risk_\rho$ is a type of normalized quantile loss metric defined as follows for $\rho \in (0, 1)$:

$$Risk_\rho(\hat{\mathbf{y}}, \mathbf{y}) = 2\frac{\sum_{i,t} QL_\rho(\hat{y}_t^{(i)}, y_t^{(i)})}{\sum_{i,t} |y_t^{(i)}|}$$

It is very popular in time series prediction papers that use the Transformer (e.g. in [1], [7]), which lead us to choose it as the metric we use in the model's evaluation. [pk: sure, but why use it specifically in our case, over pointwise metrics?] [mn: I added some details here:] Furthermore, quantile loss allows us to assess how well our model estimates series' extreme values. It is useful for applications like electricity consumption risk estimation, where one might ask how likely it is in the given time-frame that the power consumption reaches a certain level.

Chapter 3

Approximating a continuous distribution with the Transformer

While the Transformer was already proved to be a very impressive tool for the time series prediction, we believe it can be utilized even better for this purpose. We anticipate that the Transformer does not need the simplification of the time series forecasting task to the Gaussian's parameters prediction like the one used in [1]. In particular, one can apply it to model (almost) any distribution. This is made possible by leveraging Transformer's success in tasks requiring prediction of thousands categories like language translation.

This work explores whether Transformer's performance on time series prediction can be improved by training it to predict a categorical distribution for each point in the sequence instead of a Gaussian distribution.

3.1. Input serialization

3.1.1. Basic serialization

tutaj rysunek kubełeczków itd

3.1.2. Non-uniform serialization

?

3.2. Input serialization

3.3. Architecture

Chapter 4

Experiments

Chapter 5

Summary

Bibliography

[1] 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.* arXiv:1907.00235, 2019.

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

[3] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. *On the properties of neural machine translation: Encoder-decoder approaches.* arXiv preprint arXiv:1409.1259, 2014.

[4] Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. *Sharp nearby, fuzzy far away: How neural language models use context.* arXiv preprint arXiv:1805.04623, 2018.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. *Attention is all you need.* NIPS 2017.

[6] Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. *Generating wikipedia by summarizing long sequences.* arXiv preprint arXiv:1801.10198, 2018.

[7] Valentin Flunkert, David Salinas, and Jan Gasthaus. *Deepar: Probabilistic forecasting with autoregressive recurrent networks.* arXiv preprint arXiv:1704.04110, 2017.

[8] George EP Box and Gwilym M Jenkins. *Some recent advances in forecasting and control.* Journal of the Royal Statistical Society. Series C (Applied Statistics), 17(2):91–109, 1968.

[9] R. Hyndman, A. B. Koehler, J. K. Ord, and R .D. Snyder. *Forecasting with Exponential Smoothing: The State Space Approach.* Springer Series in Statistics. Springer, 2008. ISBN 9783540719182

[10] Nikolay Laptev, Jason Yosinski, Li Erran Li and Slawek Smyl. *Time-series extreme event forecasting with neural networks at Uber.* International Conference on Machine Learning.

[11] J. L. Ticknor. *A bayesian regularized artificial neural network for stock market forecasting.* Expert Systems with Applications, 40(14):5501–5506, 2013.

[12] Hongjie Chen, Ryan A. Rossi, Kanak Mahadik, Sungchul Kim and Hoda Eldardiry. *Graph Deep Factors for Forecasting.* arXiv:2010.07373.