uc3m | Universidad Carlos III de Madrid

Master's Degree in Big Data Analytics
2021-2022

Master Thesis

Attention Networks for Irregularly Sampled Time Series

Miguel Zabaleta Sarasa

Tutor: Pablo Martínez Olmos

Madrid, 2022

Abstract

The following Master's Thesis is centered around one of the most recent Machine Learning frameworks for time series: Multi-Time Attention Networks for Irregularly Sampled Time Series. We describe the necessary concepts to fully understand the methodology, explain the proposed architecture and conduct several experiments proving the flexibility and potential of the model. It is based on the corresponding paper, published at ICLR 2021.

Table of Contents

1. Introduction

In the context of analyzing data, time series are a type of data where observations have a time dependency between them. They are substantially notable in many industries: economics and finance (stock market evolution, macro-economic studies), medicine (physiological measurements across time), physics (particle dynamics) and others.

Their distinctive nature compared to other types of data (images, text, plain tabular data), along with their presence and influence in numerous fields makes them an important field of study in today's Machine Learning and Statistical Analysis.

The objective of this work is centered around presenting one of the most recent methodologies in the analysis of time series, which was presented in ICLR 2021. Multi-Time Attention Networks for Irregularly Sampled Time Series (**mTAN**) [1]. More concretely, we are interested in performing interpolation and forecasting in time series that are **multivariate**, **irregularly sampled** and **sparse**.

These time series are present in many domains. To name a few, one is health records, as time points vary depending on the physiological variable measured, and the rate of measurement is usually subjected to the patients caring routine [12]. Another field of interest where this kind of difficulty may arise is in financial data. Financial institutions are interested in the study and prediction of many different financial measures and at various time frames. This information may not be guaranteed to them as it would be confidential, which makes it is likely that only sparse, irregularly sampled time series are at their disposal.
Furthermore, the utility provided by being able to interpolate and forecast these trends may grant access to new discoveries and information about a patient (thus improving his medical diagnosis) and about the behavior of the financial system (which would increase profits)

With the intention to provide a detailed description and constructive intuition for the mentioned methodology, we firstly concentrate in explaining attention networks from a general point of view. Secondly, the Transformer is exposed, as it is the particular kind of attention network employed in the novel **mTAN**. Thirdly, we focus the discussion on variational autoencoders, as they are the training architecture which underlies the **mTAN** model.

On a second proceeding, we describe how these architectures are particularly implemented in the mentioned methodology.

Finally, the studied algorithms will be applied to real datasets, offering a description of the data, modifications in the algorithm needed to be made and presentation of results.
A last section is included for the acquired knowledge and resulting experiments, together with a brief personal note.

2. Related work

The main task we are interested in solving is analyzing multivariate, irregularly sampled, sparse time series.

In principle, this cannot be tackled by standard statistical models, as they assume fully observed, fixed-size time series (such as in the popular Box-Jenkins methodology). One could try to input the missing values as a preprocessing step (using the mean, median or another algorithm, like the SMOTE). This strategy doesn't provide good results as we are introducing a lot of artificial values since the data is sparse.
A solution with regards to the irregular distribution of time points is transforming the time points into discrete points of fixed length. This approach is very simplistic and additional processing steps are required to figure out the best way to discretize the time points. An example of such implementation can be found in Lipton et al. (2016) [2]

Given the detailed weaknesses that statistical methods seem to provide, a next idea could be to take a **machine learning** approach. There have been several models which are able to directly use an irregularly sampled time series as input. Examples of such architectures are GRUs (Chung et al. 2014 [3]), LSTMs (Neil et al. 2016 [4]) and multi-directional RNNs (Yoon et al. 2019 [5]). Although these methods do tackle the irregularly in the sampling, they struggle when the time series also is sparse or doesn't have a full set of values for each time point, which is a frequent feature in the multivariate setting.

The final and more recent field of theory is revolved around **attention mechanisms**, were the model learns to focus on particular sets of observations with the intention of capturing complex patterns. One of the most popular architectures is the **Transformer**, developed by Vaswani et al. (2017) [6].
The transformer was initially developed in the field of machine translation. Not only did it achieve better results than any of the previous state of the art model, but it had a tremendous impact in the field of AI, becoming a general-purpose architecture for Machine Learning in the next years [13]. Consequently, many industries and fields apart from NLP demonstrated that this methodology could be successfully be applied to their respective problems, each with its own nature [14].

As an illustration of the potential this model holds, one of the most popular and capable models in NLP is based on the transformer, GPT-3.
Taking this into account, there is an exceptional opportunity in correctly applying transformer networks to interpolating and forecasting multivariate, irregularly sampled, sparse time series.

With regards to this implementation, let us advance that even though the transformer was initially developed in the field of machine translation, they also introduce a positional encoding of words, which can be equated to indices of time points. However, their positional encoding is fixed.

The proposed methodology (**mTAN**) uses a Transformer, with the novelty of also learning the positional encodings and integrating the encodings with the attention module so that the model is able to learn to attend and learn complex patterns of information. This alongside with its computational

efficiency are what makes the **mTAN** a model which achieves outstanding flexibility and performs as well or better than some of the state of the art models.

3. Theoretical framework

The purpose of the following presented methodology is the analysis of multivariate, irregularly sampled, sparse time series, using attention networks.

On a first step, the preliminary concepts needed to deeply understand this architecture will be introduced. This includes attention networks, the transformer (a kind of attention network) and variational autoencoders.
Subsequently, the particular implementation that is utilized in the model of our interest is developed.

3.1. Preliminary concepts

1. Attention Networks

Attention networks were first introduced as an architecture in the field of machine translation by Bahdanau et al. (2014) [7]
The encoder-decoder family of models were the most used for such task at the time. Below is an example of the diagram of such architecture:

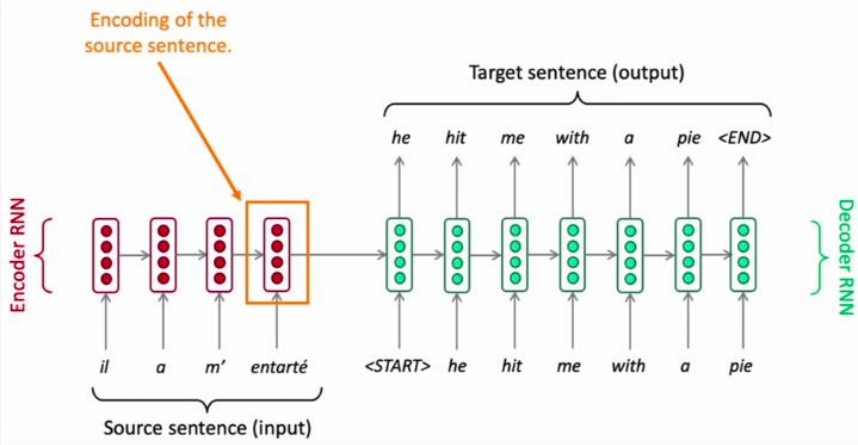


Figure 1. Encoder-decoder diagram in machine translation [8]

The encoder RNN takes the input (a sentence, in the context of machine translation) and produces an encoding (a word embedding).

It is convenient to think of embeddings/encodings as representations of a given vector, which encapsulate the most important information of such vector. Once the model is trained, the learned embeddings will provide us with the most likely translation.
In the context of time series, this framework can be used to "translate" a sparse, irregularly sampled time series into a fully-observed time series.
As can be seen through the diagram, these models encode an entire sentence into a single fixed-length vector, which is a major bottleneck in the correct encoding of the semantics of sentences.

Attention networks were invented to improve this encoding by making the embeddings received by the decoder more flexible. On each step of the decoder, the final embeddings depend not only on the last encoder embedding, but on all the other embeddings. Intuitively, different connections to the encoder will focus on a particular part of the source sentence, allowing for more flexibility and better complexity representation.

Let us now describe an attention network in more detail. It is suitable to introduce this concept from the field of *information retrieval* in a database: given a vector of *keys* that match certain *values*, we want to retrieve the most appropriate values for a given *query*, matching the query to the most similar key.
In practice, attention is a method to compute a weighted sum of the values, dependent on the similarity scores between the key and query. Different selections for these values and different ways of operating with them give rise to diverse attention networks (such as transformers, which are used in the architecture that concerns us).

Having presented the intuition behind attention, we can proceed to describe the equations that conform this architecture. The equation that mimics to obtain the most appropriate values, given the query and key-value pairs is the following:

$$attention(q, \boldsymbol{k}, \boldsymbol{v}) = \sum_i similarity(q, k_i) \times v_i$$

If the similarity function was an identity function, the weights it would provide us with would all be zero except for one case, where it would be one. Therefore, the final output would be the most fitting value. Similarly, in attention networks, the similarity function will produce a distribution over the keys, which will yield a linear combination of the values which focuses on the most similar keys to the query.
In particular, given an array of values $\boldsymbol{v}$, a query q and a set of keys $\boldsymbol{k}$, the similarity function will consist of a *softmax* function over the dot product (sometimes the scaled dot product is used) between the query and keys (called *key-query* affinities):

$$s_i = q^t k_i \text{ (key-query affinities)}$$
$$a_i = \frac{\exp(s_i)}{\sum_i \exp(s_i)} \text{ (attention weights)}$$

Finally, the attention values are computed as follows:

$$attention(q, \boldsymbol{k}, \boldsymbol{v}) = \sum_i a_i \times v_i$$

Typically, the query is the last hidden vector from the decoder, and the keys and values are both the last hidden vectors from the encoder.
This diagram represents such structure (queries in red, keys and values in blue):

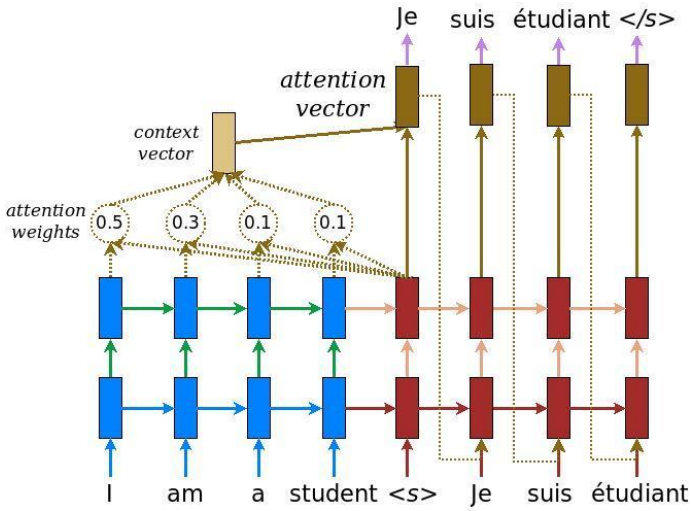


Figure 2. Attention network diagram in machine translation [9]

Now, we will introduce a particular architecture in the family of attention networks: the transformer. This will be the main model used in the multi-time attention network for analyzing irregularly sampled time series.

2. Transformers

Transformers were first proposed by Vaswani et al. (2017) [6]. They are a kind of attention network where the queries, keys and values all come from the same source (the same embedding). To illustrate this concept, below is the diagram of the model:

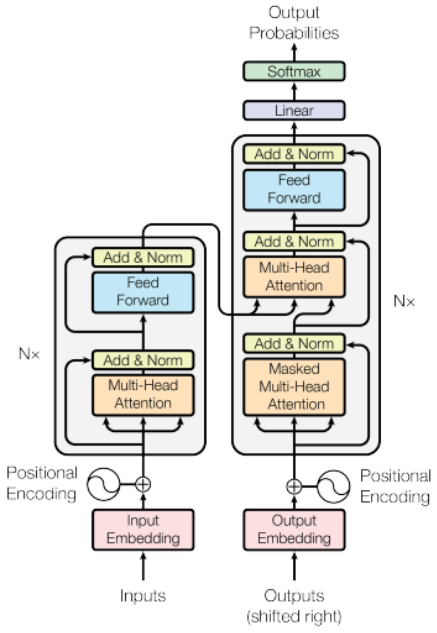


Figure 3. Architecture of the Transformer model [6]

In the previous attention network we described, the decoder and encoder were both composed of RNN networks. As we can see, these networks are not present now, and a “multi-head attention” module has replaced them.

Let us first focus on the **multi-head attention** module. Looking at the left part of the diagram, this module receives the input embedding (along with a positional encoding, explained later) and performs the attention on the embeddings directly. On the other hand, the previous attention module performed the attention on the hidden states provided by the encoder and decoder (two different embeddings). This is the reason why we indicated that transformers are a kind of attention network were the queries, keys and values all come from the same source.

Now, we will explain the multi-head attention. It is called multi-head because there are multiple layers of attention stacked upon each other. The module receives the input embeddings, and assigns these embeddings to all three keys, queries and values.
These keys, queries and values are fed to a linear transformation, with the objective of finding a better projection for them. Then, the attention computations (previously detailed) are applied using a scaled dot product. This process is done for all the heads, which are finally concatenated and aggregated linearly. Note that the attention computations for each head can be parallelized, which is one of the great advantages of the transformer.
The below figures represent the relevant diagram, along with its equations:

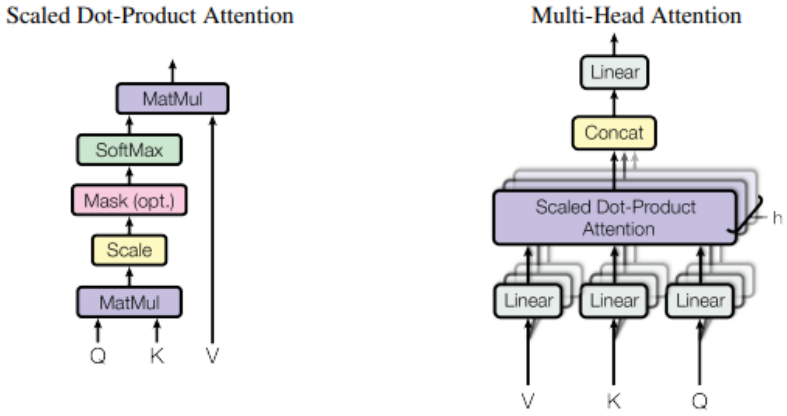


Figure 4. Single and multi-head attention diagrams [6]

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

W^O, W_i^Q, W_i^K, W_i^V are the corresponding matrices for the linear transformations used in the final output and queries, keys, values (respectively).

The multi-head attention module is the principal factor in the transformer. However, we will briefly explain the other components (Figure 3).
First, the **positional encoding**. One of the barriers that the multi-head attention module presents on its own is the **lack of ordering** in the inputs (which was lost due to the lack of an RNN). In order to solve this, a vector which indicates the position of each input is added to the keys, queries and values to preserve the order. This is the positional encoding vector, and uses sine and cosine functions of different frequencies.

The second highlight is the **feed forward network**. As one can notice, any **nonlinearities** previously induced in the RNNs have also now **dissipated**. The feed forward module ensures that nonlinearities are applied to the self-attention outputs, allowing for more flexibility in the learning.

Thirdly, we can notice the **masked multi-head attention** module. This module guarantees that the data in the **future is not utilized** when making predictions. In RNNs, there is no need to mask the data, because the next word is predicted simply based on the previous ones (Figure 2). Here, since the decoder is using the entire output embedding, we need to mask the future observations. This is achieved by including a matrix with 0's and $-\infty$'s (M) in the *softmax* function used in attention. The resulting equation is the following:

$$maskedattention(Q, K, V) = softmax\left(\frac{QK^T + M}{\sqrt{d_k}}\right)V$$

Finally, let us simply observe that the **"Add & Norm"** layers are used to accelerate the training process, by dissolving the **internal covariate shift**.

Having described the transformer (which will be used in the methodology that concerns us), we will now illustrate the Variational Autoencoder, the framework under which the model is trained.

3. Variational Autoencoders

Variational autoencoders (VAEs) belong to the area of generative modeling, where the aim is to learn a joint distribution over the given data. The primary assumptions in VAEs are captured in the following diagram:

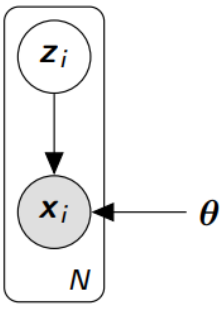


Figure 5. Assumptions in Generative Models

It is assumed that the observed data comes from an unobserved distribution of latent variables, which encode all the relevant information from the data.
θ represents the set of parameters which characterize the distribution of the data. In latent variable models, the following distributions are specified:
1. $p(\boldsymbol{z})$: prior distribution for the latent variables
2. $p(\boldsymbol{x}|\boldsymbol{z})$: likelihood of the data (given the latent variables)

We are interested in computing:
1. $p(\boldsymbol{x})$: marginal likelihood of the data
2. $p(\boldsymbol{z}|\boldsymbol{x})$: posterior distribution of latent variables

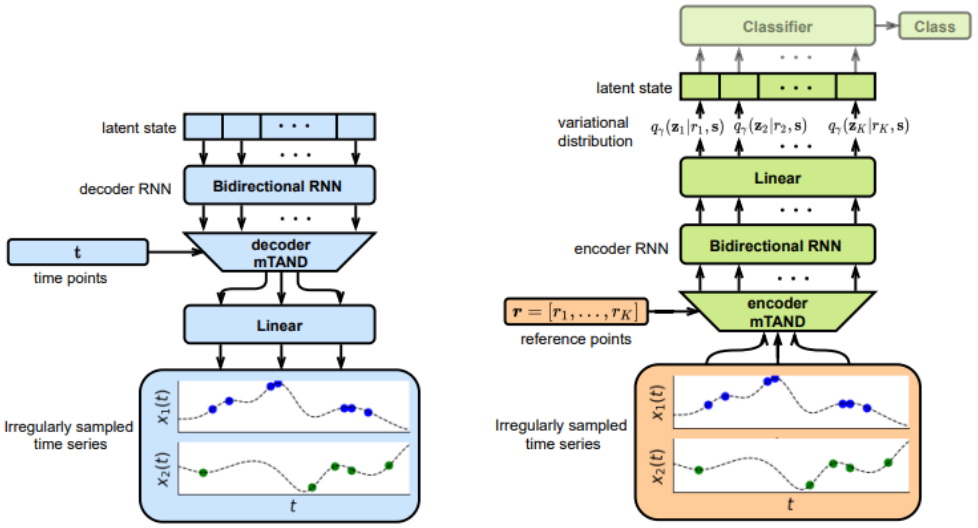
Given that the latent variables will be a low-dimensional embedding of the data, by estimating $p(\boldsymbol{z}|\boldsymbol{x})$ we are able to effectively produce a reduction in dimensionality on $\boldsymbol{x}$. This makes VAEs a probabilistic methodology to reduce the dimensionality of the data (it also includes a highly nonlinear space between $\boldsymbol{x}$ and $\boldsymbol{z}$).
The specific assumed likelihood distribution of the data is the following:

$$\boldsymbol{x}|\boldsymbol{z} \sim N\left(\mu_\theta(z), diag\left(\sigma_\theta(z)\right)\right)$$

Where $\mu_\theta(z)$, $\log \sigma_\theta(z)$ are the outputs of a neural network with parameter vector θ. This has the advantage of being a very general assumption, but it is not directly computable, as we do not know an analytical form of $p(\boldsymbol{x}|\boldsymbol{z})$.
The solution to this issue is to instead optimize a lower bound for the likelihood by approximating $p(\boldsymbol{z}|\boldsymbol{x})$ with another distribution, $q_\eta(\boldsymbol{z})$. The estimation is performed by utilizing the KL divergence, which measures the discrepancy between two distributions.
Once the parameter vectors have been optimized (η, θ), we will be able to generate any number of points from either $\boldsymbol{z}$ or $\boldsymbol{x}$, thereby creating observations from the latent space from which the data comes, or produce new data observations. The network that given $\boldsymbol{z}$, produces $\boldsymbol{x}$ is referred to as the decoding network, while the network that generates $\boldsymbol{z}$ given $\boldsymbol{x}$ is named encoder network.

3.2. Multi-Time Attention Networks

Now that we have introduced the relevant theory, we are in position to begin the description of the methodology developed in this project.
Let us recall that the objective of this architecture is to be able to operate with sparse, multivariate, irregularly sampled time series without any preprocessing. Here is the high level architecture of the model (named **mTAN-Full**):



(a) Generative Model (Decoder) (b) Inference Network (Encoder)

Figure 6. Architecture of **mTAN-Full** [1]

Essentially, the model is trained through a VAE framework. This is to say that the encoder will receive the time series and output the variational distribution, from which the latent state observations will be generated. On the other hand, the decoder will have as inputs such latent states, and from them, it will produce new data observations (i.e., an irregularly sampled time series).
Of course, the data needs to undergo some transformations in order to learn the most appropriate embeddings. The key module that will produce these embeddings is the transformer module utilized in this architecture (**mTAND**).

The principal innovation in this system is the fact that it is able to **learn** a **time representation**, while also **integrating** it correctly on an **attention** mechanism, so that the model focuses on the most relevant observations and embeddings in order to correctly learn the distribution of the data.

1. Multi-Time Attention Module

The first element that forms the multi-time attention module (**mTAND**) is a mechanism to learn a continuous time embedding. This is achieved through H embedding functions $\phi_h(t)$, where the i-th dimension is defined as follows:

$$\phi_h(t)[i] = \begin{cases} \omega_{0h} \cdot t + \alpha_{0h}, & \text{if} \quad i = 0 \\ \sin(\omega_{ih} \cdot t + \alpha_{ih}), & \text{if} \quad 0 < i < d_r \end{cases}$$

The learnable parameters are the ω_{ih}'s and α_{ih}'s.
The **mTAND** module will leverage these time embeddings into the attention network. In particular, the *queries* will be a set of reference time points and the *keys* will be assigned to the original time points in the time series. The *values* will be the time series data values.
The next equations describe the computations performed by **mTAND** to create each dimension in the embedding:

$$\text{mTAN}(t,\mathbf{s})[j] = \sum_{h=1}^{H}\sum_{d=1}^{D} \hat{x}_{hd}(t,\mathbf{s}) \cdot U_{hdj}$$

$$\hat{x}_{hd}(t,\mathbf{s}) = \sum_{i=1}^{L_d} \kappa_h(t,t_{id})\, x_{id}$$

$$\kappa_h(t,t_{id}) = \frac{\exp\left(\phi_h(t)\mathbf{w}\mathbf{v}^T\phi_h(t_{id})^T/\sqrt{d_k}\right)}{\sum_{i'=1}^{L_d}\exp\left(\phi_h(t)\mathbf{w}\mathbf{v}^T\phi_h(t_{i'd})^T/\sqrt{d_k}\right)}$$

It is appropriate to now recall the attention function. It was defined as follows:

$$attention(Q,K,V) = softmax\big(similarity(Q,K)\big)V$$

We can first realize that the third equation is in reality a *softmax* function applied to some expression. Disaggregating such expression, we can see it is the scaled inner product between the time embeddings applied to the reference time points (*queries*) and the original time points (*keys*). On the second equation, the attention values are computed.
The first equation projects the attention outputs into a linear space.

It is also worth noting that there are H time embeddings functions utilized to produce the attention outputs. Additionally, the final linear projection is done through a matrix which depends on the time embedding (h), the time series dimension (d) and the final embedding dimension (j). This makes the **mTAND** module extremely flexible and able to correctly make use of all the time embeddings, thus learning complex time patterns from sparse, irregularly sampled time series.
The below diagram provides the recently detailed architecture:

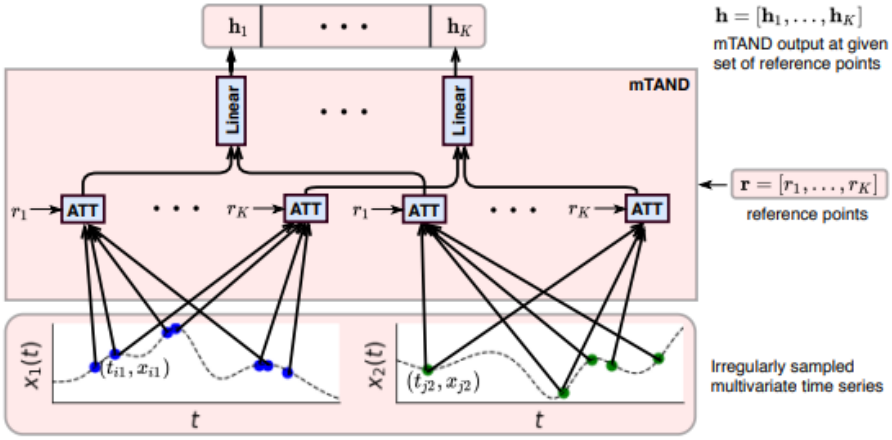


Figure 7. Architecture of the **mTAND** module [1]

2. Encoder-Decoder Framework

The encoder-decoder framework describes the architecture of the entire model. It is developed within the VAE framework. The diagram that represents such architecture is illustrated in Figure 6.

We begin by describing the encoder component. Let us recall that the objective of the encoder in the VAE framework is to estimate the distribution of latent states. Once they are estimated, the decoder will use this distribution to approximate the data distribution, thus generating new observations.

As far as how the **interpolation** is performed, the **encoder mTAND** network begins by receiving the reference time points (time embeddings on these points will be used as queries) along with the original time series. Importantly, the original time series is **subsampled**. Once the attention is performed, the output is transformed through a RNN network so as to induce temporal order in the embeddings, much similarly as was done in the transformer with the positional encodings.
A series of linear transformations are applied to the resulting embeddings, projecting them into the space of latent variables. Therefore, the original values in the time series are effectively transformed into a series of observations in the latent space.

The second part of the architecture concerns the **decoder**. It is quite close to an inverse counterpart of the encoder. Firstly, it receives the latent state observations and transforms them through a RNN to integrate temporal dependencies on them. Then, the attention is performed to these values. Now, the **whole set** of original **time points** are retrieved back. The queries on the attention module will be the time embeddings applied to this set of time points, while the keys will be formed by the time embeddings of the queries.
After the attention values are computed, they are again projected linearly into the space of the multivariate time series.

The fact that the encoder uses a subset of the original time series, while the decoder utilizes the entire set of time points is the **key concept** that enables **interpolation** and **forecasting**. The result of this process is that the time series generated by the encoder will be an interpolation over the time series subset. Additionally, the loss function is evaluated on the whole time series thus the system learns to interpolate out of a subset of the times series. In conclusion, once the model is trained, we can load a sparse, irregularly sampled time series along with a set of temporal indices from which we do not have observations, and the system will be able to predict the values on those unobserved indices. Predicting on past indices will constitute **interpolation**, while predicting on future indices will account for **forecasting**.
The model is also capable of **classification**. The only change in the architecture is the addition of a classifier module (the paper used a GRU network) which receives the latent states observations and predicts the target variable.

4. Applications

We begin the applications section with an applied example of the algorithms we have previously illustrated, using real data.

4.1 Software

As far as the resources with which this project has been developed, it is worth noting that the computer used was a laptop HP EliteBook 840 G5 with a processor Intel(R) Core(TM) i5-8250U (CPU @ 1.60GHz 1.80 GHz) and 7.85 GB of usable RAM.

4.2 Experiments

The code is available in the following GitHub repository.

1. Data description

Tufts fNIRS to Mental Workload Dataset

This dataset consists of functional near-infrared spectroscopy (fNIRS) measuring the brain activity of 68 participants over one 30-60 minute experimental session. At each time step, 8 measurements are recorded according to the combinations of different medical situations, (2 hemoglobin concentration changes, 2 optical data types and 2 spatial locations of the device of measurement). All measurements are recorded at a sampling rate of 5.2 Hz (reference for this dataset can be found in [10]).
The particular processed dataset used in our experiments is measured with a sliding window of 30 seconds and 0.6 seconds stride, which makes up for 1487 time points for each patient.

Financial data

The second dataset used in our experiments is composed of stock market data of some of the biggest tech companies in the world: Apple, Amazon, Facebook, Google, Microsoft, Netflix and Tesla (available at [11]).
There are 4 time series per company, with time steps measurements in 1 min, 5 min, 30 min 1 hour respectively, and each time series has 4 variables (open price, highest price traded, lowest price traded, number of shares traded). In total, there are 28 time series, where the earliest time point is 02-08-2021 at 04:00:00 and the furthest is 12-08-2021 at 19:59:00.

2. Implementation details

Tufts fNIRS to Mental Workload Dataset

For data preprocessing, we took the average values for each time step and randomly sampled 50-90% of the values in each univariate variable so that the final data is sparse and irregularly sampled (as suggested in [1]).
The split between training and test set was 80% and 20% respectively.

For hyperparameter optimization, we trained the model for 50 iterations, using batches of size 10 and 0.001 learning rate. We evaluated the MSE on present data (interpolation) and future observations

(forecast), both on the training and test sets. The hyperparameters that we tuned were the number of reference points and the number of time embedding functions.
These are the results obtained during training:

Reference points	Embedding functions	Interpolation	Forecasting
8	1	0.0141	0.999
8	2	0.0154	0.965
8	4	0.0170	1.050
16	1	0.0132	1.019
16	2	0.0133	1.036

The best results were provided using 16 reference points and 2 embedding functions.

As for the other hyperparameters, their values correspond to the default/optimized values implemented in the experiments carried out in [1]. These are the details:

- Optimized parameters
    - # dimensions in attention embedding: 32 encoder, 50 decoder
    - # observations used for stabilizing VAE training: 5
- Default parameters
    - # dimensions in variational distribution: 32

Financial data

The only additional preprocessing step needed on this dataset was to transform the date times into indices. We considered the earliest date as index 1, and each step in the time index be of length 1 minute. Training batches of size 4 were used, and the rest of the preprocessing pipeline is similar as the one applied in the fNIRS dataset.

The parameter values also remained the same, while the set of tuned hyperparameters was carried out for 50 iterations, evaluating the performance of number of reference points and number of time embedding functions for the following combinations (the rest didn't fit into memory):

Reference points	Embedding functions	Interpolation	Forecasting
8	1	0.0160	0.1640
8	2	0.0101	0.1653
16	1	0.0029	0.1553

The best performance was achieved with 16 reference points and 1 embedding function.

3. Final results

After tuning the hyperparameters, we re-trained the models for 200 iterations, achieving the following results:

		Interpolation	Forecasting
fNIRS	Training set	0.0130	1.0144
	Test set	0.0158	1.0119
Financial	Training set	0.0023	0.1879
	Test set	0.0086	0.1768

In the light of these results the model seems to perform quite well, specially taking into consideration that there are many hyperparameters which we did not optimize. Additionally, it is worth noting that the model is able to achieve a good performance on a very low number of observations (68 for fNIRS, 28 for financial).
As a point of comparison, the authors of the paper in **mTAN** trained the model with thousands of time series, and tuned most of the hyperparameters with a variety of values.

5. Conclusions

We have presented and described in detail one of the most recent methodologies in the field of time series analysis, illustrating the machine learning architectures and frameworks underlying it and applying the acquired knowledge to real datasets.

Even though the obtained results seem promising, there are some obvious improvements that could have been made. One of which is the mentioned increase of the training dataset and extended hyperparameter tuning phase.
Furthermore, the authors in **mTAN** suggest that instead of using recurrent networks, convolutional networks could provide potentially better performance, due to the improved parallelism that these architectures have over recurrent neural networks.

I would like to conclude this work with a personal note, in which I express my satisfaction for the developed theory, the final results obtained and the effort it has undertaken. For me, having expanded my knowledge of Machine Learning in this way has been a wonderfully fulfilling experience, studying subjects such as Attention Networks and, as an outstanding point, deepening my knowledge in one of the most relevant models in recent Machine Learning, the Transformer. I would also like to send my appreciation to my supervisor for his opportune considerations and supervision.

References

[1] Shukla, S. N. et al. Multi-time attention networks for irregularly sampled time series. 2021

[2] Zachary C Lipton et al. Directly modeling missing data in sequences with RNNs: Improved classification of clinical time series. In Machine Learning for Healthcare Conference, pp. 253–270, 2016

[3] Junyoung Chung, et al. Empirical evaluation of gated recurrent neural networks on sequence modeling. 2014

[4] Daniel Neil et al. Phased lstm: Accelerating recurrent network training for long or event-based sequences. 2016

[5] J. Yoon et al. Estimating missing data in temporal data streams using multi-directional recurrent neural networks. 2019

[6] Vaswani, A et al. Attention is all you need. 2017

[7] Bahdanau, D. et al. Neural machine translation by jointly learning to align and translate. 2014

[8] Stanford CS224N NLP with Deep Learning. Winter 2021, Lecture 7. Course link

[9] https://www.tensorflow.org/text/tutorials/nmt_with_attention

[10] https://tufts-hci-lab.github.io/code_and_datasets/fNIRS2MW.html

[11] https://firstratedata.com/

[12] Li, S. C. X et al. Learning from irregularly-sampled time series: A missing data perspective. 2020

[13] “State of AI Report: Transformers Are Taking the AI World by Storm”. https://exchange.scale.com/public/blogs/state-of-ai-report-2021-transformers-taking-ai-world-by-storm-nathan-benaich

[14] “Transformer neural networks are shaking up AI”. https://www.techtarget.com/searchenterpriseai/feature/Transformer-neural-networks-are-shaking-up-AI