Permutation-Invariant Neural Networks and Applications for Irregularly-shaped Time Series

Hava Chaptoukaev
Supervisor: Pierre-Alexandre Mattei
Co-supervisor: Hugo Schmutz

Université Côte d'Azur

Abstract. Permutation-invariant neural networks are deep neural networks whose output is invariant to permutations of the input. Such networks have received considerable attention recently, notably through the influential *Deep Sets* paper [1]. The goals of this project would be to first understand and implement the Deep Sets architecture, and try to use it to model time series. In particular, one of the advantages of using permutation-invariant architectures for time series is to be able to handle irregularly-sampled series, as demonstrated by the *Set Functions for Time Series* paper [2].

Keywords: Permutation-Invariance · Set Functions · Neural Networks · Irregularly-Sampled Time Series.

1 Introduction

A time series is a sequence of data points indexed in time order and most commonly taken at equally spaced points in time. Regular time series analysis and forecasting techniques such as Recurrent Neural Networks (RNN) and similar architectures usually assume that the measurements in a sequence are taken at regular intervals, and if several measurements are taken they are aligned. In reality irregularly-sampled times series naturally occur in many real world domains, ranging from natural disasters typically occurring at irregular time intervals, to healthcare where measurements are not necessarily observed at a regular rate. Indeed, in the latter example a patients' state of health is usually observed at irregular intervals and different variables are observed at different times, which can be challenging to model as the available data does not yield the fixed-dimensional representation that most standard machine learning models require. An irregularly-sampled time series can be described as a sequence of time-value pairs with non-uniform intervals between successive time points. More precisely, univariate irregularly-shaped time series are generally characterized by variable time intervals between observations, while multivariate irregularly-sampled time series can also have aligned or unaligned observation times across dimensions, as shown in Fig. 1. Such data present challenges to many classical models from machine learning and statistics. However, there has been significant progress on developing specialized models and architectures for learning

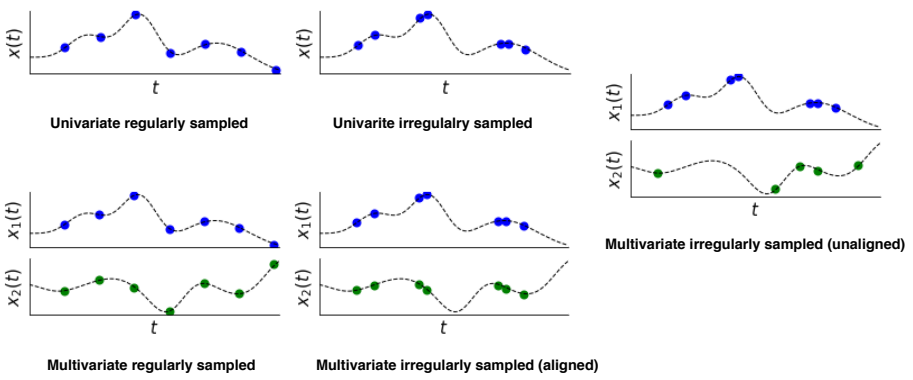

Fig. 1. Illustration of regularly and irregularly-sampled univariate and two-dimensional multivariate time series. (Figure taken from [3].)

from irregularly-sampled time series data in recent work, and Shulkla and Marlin [3] surveyed and classified the different approaches in the domain including the data representations they are based on. In this paper we first present a brief survey of these approaches, to then focus on permutation-invariant models.

2 Related Work

Some common approaches to analyze unevenly spaced time series consist in transforming the data into equally spaced observations using interpolations as shown in [4] and [5], or data imputation. However, transforming data in such ways can introduce a significant number of biases [6]. Moreover, the presence or absence of a measurement and its observation may carry information of its own, meaning that imputing the missing values is not always desired. Most of the recent work in irregularly-shaped time series handling aim to overcome these problems, and can be categorized under the following categories.

Discretization Methods Irregular sampling in time series is closely related to the problem of classification in the presence of missing data. While some works approached the problem of irregularly-sampled time series from a missing data perspective, other frameworks were specifically designed to support it. For example Bahadori and Lipton [7] converted an irregularly-sampled time series into a missing data problem by discretizing the time axis into non-overlapping intervals, and declaring intervals in which no data was sampled as missing.

Interpolation Methods Other methods consisting in the learning of imputation schemes also exist. The frameworks of this type are usually composed of an

imputation scheme and a classifier both trained end-to-end using gradient-based training. Li and Marlin [8] introduced a deterministic interpolation framework based on variational auto-encoders (VAE) and generative adversarial networks (GAN) to model the distribution of irregularly-sampled time series. Similarly Shukla and Marlin [9] proposed an imputation scheme based on the use of a semi-parametric interpolation network followed by the application of a prediction network. The parameters of the interpolation network are trained with the prediction network in an end-to-end setup. Among other noticeable approaches, methods based on probabilistic interpolation have been proposed. For instance, Li and Marlin [10] proposed the Gaussian Process Adapters, a scalable uncertaintly-aware framework for classification of sparse and irregularly-sampled time series, where the parameters of a Gaussian Process Kernel are trained in combination with a number of black-box classifiers learnable using gradient descent. This approach was further extended to multivariate time series by Futoma et al. [11] using Multi-task Gaussian Processes [12], and making the approach more compatible with time series of different lengths by applying a Long Short Term Memory (LSTM) classifier. In contrast, Lu et al. [13] proposed a similarity based interpolation framework. The authors introduced a non-parametric distance measure for time series by representing each irregularly-sampled time series in a reproducing kernel Hilbert space with a kernel learned from the data, allowing to build kernel-based classifiers or regression schemes supporting missing data. Along these lines, Li and Marlin [14] introduced a kernel-based framework allowing the comparison and classification of irregularly-sampled time series, by representing each time series through the Gaussian process posterior it induces under a Gaussian process regression.

Recurrence-based Methods In other recent work, Che et al. [15] presented the GRU-D network, a variant of the state of the art recurrent network Gated Recurrent Unit (GRU), exploiting the missing patterns in the data to build imputation schemes. Rubanova et al. [16] proposed latent ordinary differential equation (ODE) models as a natural way to model continuous-time hidden dynamics defined by ordinary differential equations. Latent ODEs extend the neural ODE models [17], which enables modeling of complex ODEs using neural networks.

3 Background

Many popular machine learning models, including neural networks and Gaussian processes require fixed-dimensional data inputs such as vectors, matrices or tensors. Their extension to handle unstructured set inputs is not trivial, although a range of machine learning problems such as statistic estimation, point cloud classification or outlier detection, can be naturally formulated in terms of sets. The concept of sum decomposition, attention mechanisms and the notion of exchangeability are some key elements in the process of extending well-known standard methods to set inputs.

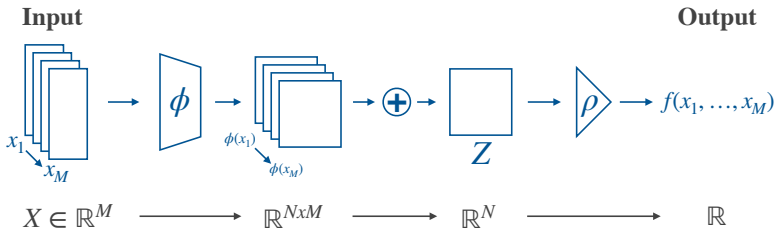

Fig. 2. Illustration of the model structure proposed by Zaheer et al. [1]. The sum operation enforces permutation invariance for the model as a whole. φ and ρ can be implemented by neural networks. (Figure taken from [19].)

3.1 Permutation-Invariant Neural Networks

Models for set inputs have gained attention recently. For instance, Zaheer et al. [1] considered objective functions defined on sets that are *invariant to permutations* in the paper *Deep Sets*. Precisely, [1] characterizes and provides a family of permutations-invariant functions of a special structure allowing the design of a deep network operating on sets. Similarly, Ravanbakhsh et al. [18] used a permutation-invariant network for point-cloud classification and introduced a *permutation-equivariant* layer for deep learning set structures.

Permutation-Invariance A function f that transforms its domain X into its range Y is permutation-invariant if it is indifferent to the ordering of its input elements. Many authors [1, 18] enforced the property of permutation-invariance using the notion of *sum decomposition* as illustrated in Fig. 2. The following theorem characterizes the structure of such functions in the case where the input is a countable set.

Theorem 1. *A function f operating on a set X having elements in a countable universe, is a valid set function iff there are functions $\varphi : R \to Z$ and $\rho : Z \to R$ such that*

$$f(X) = \rho\Big(\sum_{x \in X} \varphi(x)\Big) \tag{1}$$

In other words, f is invariant to the permutations of its input, if it is sum-decomposable via a latent space Z. Summation being permutation-invariant, a sum decomposition is also permutation-invariant.

Invariant Architecture Based on the previous theorem, Zaheer et al. [1] proposed the *Deep Sets* architecture, as a general strategy for inference over sets. In a first step, each observation x_j of an input set X is transformed, possibly through several layers, into some representation $\phi(x_j)$. The representations $\phi(x_j)$ are then added up and the output is processed using the ρ network as in any

deep network, e.g. using fully connected layers. However, representing functions on sets has limitations, Wagstaf et al. [19] show how considering continuity on uncountable domains such as $\mathbb{R}$ is necessary and show that an implementation of theses models via neural networks can only be achieved with a latent dimension at least the size of the maximum number of input elements. In other recent work, Murphy et al. [20] presented a permutation-invariant pooling layer for variable sized inputs, as the average of a permutation-sensitive function applied to all the possible orderings of the input sequence.

3.2 Attention-based Neural Networks

Models based on *attention* extend the previous concept by performing a weighted summation of a set features. For example, Vaswani et al. [21] introduced the *Transformer* network, an encoder-decoder structured architecture based solely on attention mechanisms and using stacked self-attention and fully connected layers for both the encoder and decoder. In [21] the authors simply describe attention as a function mapping a query and a set of key-value pairs to an output, and introduce the *scaled dot-product attention* and the *multi-head attention*.

Scaled Dot-product Attention The input of the scaled dot-product attention consists of queries and keys of dimension d_k , and values of dimension d_v, packed together into matrices Q, K and V respectively. In practice, the attention function computes the dot product of the query with all keys, scales it, and applies a softmax function to obtain weights to apply on the values.

Multi-Head Attention The multi-head attention on the other hand, consists of several scaled dot-product attention layers running in parallel and concatenated afterwards. Instead of computing a single attention function with d_{model} dimensional keys, values and queries, this method first projects Q, K, V onto h different d_q, d_k, and d_v $(= d_k)$ dimensional vectors. An attention function is then applied to each of these h projections, that are then concatenated and once again projected. The projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_q}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$. The output of the multi-head attention is a linear transformation of the concatenation of all attention outputs resulting in the final values

$$m = [a_1, \ldots, a_h]W^O \tag{2}$$

where the elements a_i are the self attention layers computed for each head i and are given by

$$a_i = \frac{\exp(e_i)}{\sum_j \exp(e_j)} V W_i^V \quad \text{where } e_i = \frac{QW_i^Q.(KW_i^K)^T}{\sqrt{d_k}} \tag{3}$$

Based on [21], Lee et al. [22] presented the *Set Transformer*, an attention-based neural network module specifically designed to model interactions among elements of the input set. Similarly to the *Transformer* architecture, the *Set Transformer* consists of an encoder followed by a decoder, but a distinguishing feature is that each layer in the encoder and decoder attends to their inputs to produce activations.

3.3 Partially Exchangeable Networks

In other recent work, Wiqvist et al. [23] introduced Partially Exchangeable Networks, a family of deep neural networks invariant to block-switch transformations. Exchangeability being the simplest form of permutation invariance, and partial exchangeability being a weaker notion of invariance, [23] states the *Deep Sets* architecture [1] can be generalized to partial exchangeability. The condition of exchangeability was also exploited by Garnelo et al. [24, 25] to model the joint distributions of the values the random variables in a stochastic process take, by computing a global latent variable via summation. Bloem-Reddy and Teh [26] have provided a detailed overview of invariant neural networks, and have characterized their structure.

4 Set Functions for Time Series Classification

One of the advantages of using permutation-invariant architectures for time series analysis is to be able to handle the problem of classification of irregularly-sampled and unaligned time series. Horn et al. [2] extended recent advances in set function learning to these particular tasks in the paper *Set Functions for Time Series*. The approach is inspired by [1], [21] and [22], and the proposition behind the *Set Functions for Time Series* model (SEFT) is to rephrase the problem of classifying time series as classifying a set of observations.

4.1 Notations

Horn et al. [2] describe a D-dimensional time series as a set S_i of $M := |S_i|$ observations s_j such that $S_i := \{s_1, \ldots, s_M\}$, where each observation s_j is represented as a tuple (t_j, z_j, m_j) consisting of a time value $t_j \in \mathbb{R}^+$, an observed value $z_j \in \mathbb{R}$, and a modality indicator $m_j \in \{1 \ldots D\}$. Thus representing the whole time series as

$$S_i := \{(t_1, z_1, m_1), \ldots, (t_M, z_M, m_M)\}. \tag{4}$$

By doing so, the problem of classifying irregular time series can be phrased as learning a function on a set of many time series observations. In addition, this definition actually allows the length of each time series to differ, and does not require for the time series to be aligned, i.e. sampled at the same time. All time series and their corresponding labels are collected in data set $\mathcal{D}$. A dataset $\mathcal{D}$ consisting of n time series can be described as $\mathcal{D} := \{(S_1, y_1), \ldots, (S_n, y_n)\}$, where $y_i \in \{1, \ldots, C\}$ is the class label of each time series.

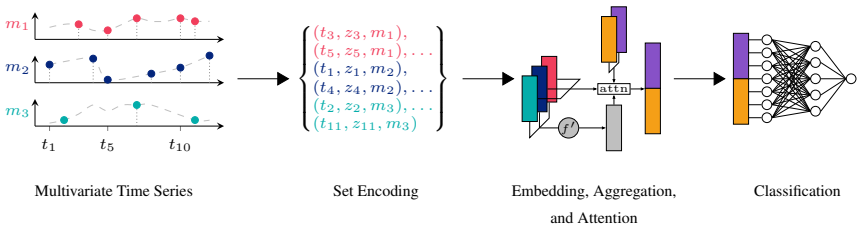

Fig. 3. Illustration of the SEFT architecture [2]. An example of potential input is encoded as tuples consisting of a time, a value and a modality indicator. All observations are summarized as sets of such tuples and the elements of each set are summarized using a set function f'. An attention mechanism is applied on the summarized representation and on the individual set elements to learn the importance of individual observations. Finally, results of each attention head, illustrated in purple and orange blocks, are then concatenated and used as the input for the final classification layers. (Figure taken from [2].)

4.2 Time Encoding

Inspired by [21], the time point of an observation is represented on a normalized scale using a variant of *positional encoding*. The time encoding converts the 1-dimensional time axis into a multi-dimensional input by passing the time t of each observation through multiple trigonometric functions, thus allowing to representing an observation as $s_j = (x(t_j), z_j, m_j)$, where

$$\begin{aligned} x_{2k}(t) &:= \sin\left(\frac{t}{t_{max}^{2k/\tau}}\right) \\ x_{2k+1}(t) &:= \cos\left(\frac{t}{t_{max}^{2k/\tau}}\right) \end{aligned} \tag{5}$$

with $\tau \in \mathbb{N}^+$ the dimensionality of the time encoding, $k \in \{0, \ldots, \frac{\tau}{2}\}$, and t_{max} the maximum time scale expected in the data.

4.3 Model Overview

An overview of SEFT's architecture is schematized in Fig. 3. Similarly to *Transformers* [21], the used approach is based on scaled dot-product attention with multiple-head in order to cover the different aspects of the aggregated set. But while *Transformer* blocks use the information from all set elements to compute the embedding of an individual set element, the SEFT architecture computes the embeddings of set elements independently. This aspect is particularly relevant as the set elements are individual observations of a time series in this case. The SEFT model is based on the *Deep Sets* framework [1], and amounts to computing multivariate data set specific summary statistics optimized to maximize

classification performance, using a set function f of the form

$$f(S) = \rho\left(\frac{1}{|S|}\sum_{s_j \in S}\phi(s_j)\right) \tag{6}$$

where $\phi : \Omega \to \mathbb{R}^d$ and $\rho : \mathbb{R}^d \to \mathbb{R}^C$ are neural networks, $d \in \mathbb{N}^+$ is the dimensionality of the latent representation, and s_j is a single observation of the time series S. In addition, it is also shown that an universal function representation f can only be guaranteed for $d \geq max_i|S_i|$. The method allows encoding sets of arbitrary sizes into a fixed representation, however the *mean* aggregation being particularly sensitive to extreme values, many irrelevant observations could influence the result of the set function, as the importance of an observation shrinks proportionally to the size of the set. To overcome this problem, the idea is to use a *weighted mean* to allow the model to decide which information is relevant. To do so, the model uses a scaled dot-product attention with multiple heads $i \in \{1, \ldots, m\}$ to compute the weight of an individual time series,denoted $a(S, s_j)$, that depends on the whole set of observations S and the value of one set element s_j.

4.4 Attention-based Weighted Sum

To compute the attention weight function of an individual time series, an embedding of the set elements is first computed using a smaller f' set function, and the concatenation of this set representation with the individual set element is then projected into a d-dimensional space. The projections are collected in a matrix K such that

$$K_{j,i} = [f'(S), s_j]^T W_i \tag{7}$$

where $W_i \in \mathbb{R}^{(im(f')+|s_j|)}$ is a projection parameter matrix, $K \in \mathbb{R}^{|S|\times d}$, and $f'(S) = \rho(\frac{1}{|S|}\sum_{s_j \in S}\psi(s_j))$ is an unweighted set function. We can notice that computing $f'(S)$ is equivalent to applying the *Deep Sets* model using a *mean* aggregation. In addition, query points are collected in a matrix $Q \in \mathbb{R}^{m\times d}$ where each row Q_i is specific to an attention head and allows the head to focus on individual aspects of a time series. The model can then summarize different aspects of the data set and compute weights via

$$a_{j,i} = \frac{\exp(e_{j,i})}{\sum_j \exp(e_{j,i})} \quad \text{with } e_{j,i} = \frac{K_{j,i}Q_i}{\sqrt{d}} \tag{8}$$

where $a_{j,i}$ represents the amount of attention that head i gives to set element j. For each head i, the individual set elements are transformed into representations $\phi(s_j)$, multiplied with the attentions derived for the individual elements, and averaged such that

$$r_i = \sum_j a_{j,i}\phi(s_j) \tag{9}$$

The computed representations are then concatenated, i.e. $r = [r_1, \ldots, r_m]$, and passed to the ρ network to be aggregated. The final output of the set function f is given by

$$f(S) = \rho\Big(\Big[\sum_j a_{j,1}\phi(s_j), \ldots, \sum_j a_{j,m}\phi(s_j)\Big]\Big) \tag{10}$$

In the model setup, the matrix Q is initialized with zeros such that the beginning of the training is equivalent to training without weights.

4.5 Loss Function

In the implementation the functions ϕ and ρ are chosen to be *multilayer perceptron* neural networks parametrized by weights θ and σ respectively. The loss function optimized during the training is overall defined as

$$\mathcal{L}(\theta, \sigma) := \mathbb{E}_{(S,y)\in\mathcal{D}}\Big[\ell\Big(y; \rho_\sigma\Big(\sum_{s_j \in s} a(S, s_j)\phi_\theta(s_j)\Big)\Big)\Big] \tag{11}$$

where $\ell(.)$ is the cross-entropy loss.

5 Experiments and Results

The goal of this project is to understand and implement permutation-invariant neural networks, and try to use them to model time series. More precisely, we would like to train such a model to perform classification on times series.

5.1 Experiment Proposal

We are interested in applying the methodology used in [2] to the Deep Sets [1] architecture. The idea is to encode each individual observation of a time series as tuples consisting of a time, a value and a modality indicator similarly to [2], allowing to represent a time series as a set of such tuples. Inspired by [2], we also embed the time value of each observations using a relative time encoding. The elements of each sets are then summarized using a set function. We implement a simple network with 4 hidden fully connected layers for both ϕ and ρ. We use a cross-entropy loss in combination with a softmax activation function in the last layer of ρ for classification. The model is trained for 30 epochs using an Adam optimizer with a batch size of 1, and an initial learning rate of 1×10^{-3}.

5.2 Synthetic Data

We first test our model on simple synthetic data. We begin by generating data sets of univariate or multivariate times series of t measurements with random trends and random periods, as illustrated by Fig. 4. To reflect the structure of irregularly-shaped time series, we then randomly select t_m unevenly spaced

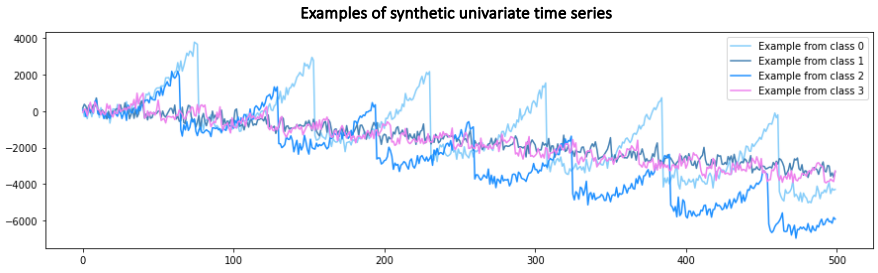

Fig. 4. We generate a data set containing univariate times series from 4 different classes, and where classes are visibly distinct enough to allow the model to easily discriminate between the labels.

and unaligned measurements for each instance, and train our network to perform classification. On such data, the model is able to easily classify both univariate or multivariate time series, and can reach an accuracy of 100% in under 30 epochs. However, the complexity of the task and the convergence speed entirely depends on the nature of the generated data. For instance, Fig. 5 illustrates the impact the number of random measurements t_m we select has on the performance of the model for a fixed number of classes to predict. It is also worth mentioning that the positional encoding of the time values has no real effect on such simple data, as the model performs as well with as without it in this context.

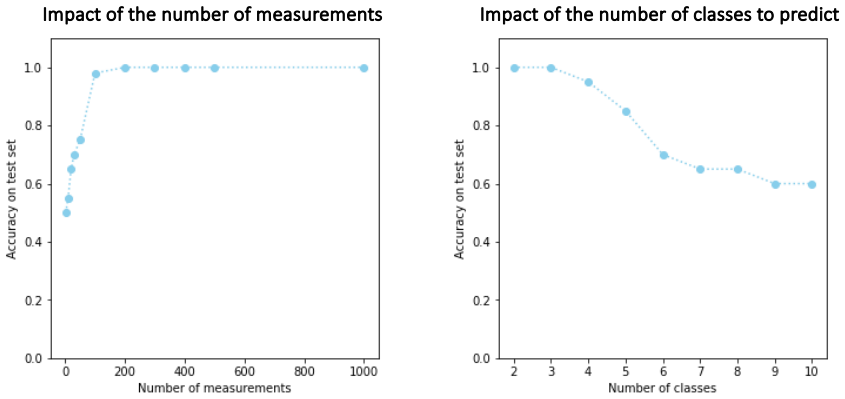

Fig. 5. After 10 epochs. (Left) Evolution of the test accuracy w.r.t. the number of measurements t_m kept in the times series, for a fixed number of classes $c = 3$ and a fixed $t = 1000$. (Right) Evolution of the test accuracy w.r.t. the number of classes to predict, for a fixed $t = 1000$ and a fixed $t_m = 500$.

5.3 Physionet 2012 Mortality Prediction Challenge

We then apply the model to real data. We selected the 2012 Physionet challenge data set in order to compare our results with the results in the SEFT paper [2] which is unfortunately lacking an ablation study that could highlight the importance of the attention mechanisms.

Description of the Data Set The 2012 Physionet challenge data set [27] is a data set with irregularly-sampled and non-synchronized measurements, and composed of 12,000 ICU stays each of which lasts at least 48h. Depending on the course of the stay and patient status, up to 37 time series variables were measured (e.g. blood pressure, lactate, and respiration rate). While some modalities might be measured in regular time intervals (e.g. hourly or daily), some are only collected when required; moreover, not all variables are available for each stay.

Performance of the Classifier We are focusing on the task of predicting patient mortality. The data being highly imbalanced and with a prevalence of around 14%, our model performs poorly when tested on the whole data set. The network is unable to learn and only outputs the majority class. However we train the model once again on a subset of the original data set that we artificially balance, to evaluate the performance of the Deep Sets architecture applied to time series classification. In this second scenario, our model reaches an accuracy of 70.1% after 30 epochs, and using a dimensionality $\tau = 10$ for the embedding of the time values. This experiment also particularly highlights the importance of the time encoding, as without it the model does not converge and cannot exceed 50% accuracy (i.e. random).

6 Conclusion

In this work, we surveyed the different approaches existing for the analysis of irregularly-sampled times series, and focused particularly on attention-based and permutation invariant models. We set up a simple experiment to model and classify time series using a network of the Deep Sets architecture. While the model shows impressive results on synthetic time series, it has some limitations on more complex data. Firstly, we observed that it is not able to deal with imbalanced data sets. Secondly, there persists a gap between our model, and models based on attention. For reference, while we reach an accuracy of 70.1% on the Physionet 2012 data set, the SEFT [2] and the Transformer [21] models can respectively achieve 75.3% and 83.7%. Extending our model by adding attention mechanisms in the network could allow it to decide which observations are relevant and which should be considered irrelevant, and should be done in future work. Further leads could as well include investigating the relationship between attention mechanisms and interpretability of such models, or improving the robustness of attention computation so that elements with low attention values do not get neglected.

References

1. *Deep Sets*, M. Zaheer, S. Kottur, S. Ravanbhakhsh, B. Póczos, R. Salakhutdinov, A. Smola in *NeurIPS*, 2017.
2. *Set Functions for Time Series* M. Horn, M. Moor C. Bock, B Rieck, K Borgwardt in *ICML*, 2020.
3. *A Survey on Principles, Models and Methods for Learning from irregularly-sampled Time Series: From Discretization to Attention and Invariance*, S. N. Shukla, B. M. Marlin, *preprint*, 2020, arXiv:2012.00168.
4. *Interpolation of irregularly-sampled data series*, H. M. Adorf in *Astronomical Data Analysis Software and Systems IV*, 1995.
5. *Statictical models for unequally spaced time series*, A. Beygelzimer, E. Erdogan, S. Ma, I. Rish in *proceedings of the 2005 SIAM International conference on data mining*, 2005.
6. *A Framework for the Analysis of Unevenly Spaced Time Series Data*, A. Eckner, *preprint*, 2012.
7. *Temporal-Clustering Invariance in Irregular Healthcare Time Series*, M.T. Bahadori, Z.C. Lipton, *preprint*, 2019, arXiv:1904.12206.
8. *Learning from Irregularly-Sampled Time Series: A Missing Data Perspective*, S.C-X. Li, B.M. Marlin in *ICML*, 2020.
9. *Interpolation-Prediction Networks for irregularly-sampled Time Series*, S.N. Shukla, B.M. Marlin, *preprint*, 2019, arXiv:1909.07782.
10. *A scalable end-to-end Gaussian process adapter for irregularly-sampled time series classification*, S.C-X. Li, B.M. Marlin in *NeurIPS*, 2016.
11. *Learning to Detect Sepsis with a Multitask Gaussian Process RNN Classifier*, J. Futoma, S. Hariharan, K. Heller, *preprint*, 2017, arXiv:1706.04152.
12. *Multi-task Gaussian Process Prediction*, E.V. Bonilla, K-M. A. Chai, C.K.I. Williams in *NeurIPS*, 2008.
13. *A reproducing kernel hilbert space framework for pairwise time series distances*, Z. Lu, T.K. Leen, Y. Huang, D. Erdogmus, in *Proceedings of the 25th ICML*, 2008.
14. *Classification of Sparse and irregularly-sampled Time Series with Mixtures of Expected Gaussian Kernels and Random Features*, S.C-X. Li, B.M. Marlin in *UAI*, 2015.
15. *Recurrent Neural Networks for Multivariate Time Series with Missing Values*, Z. Che, S. Purushotham, K. Cho, D. Sontag, Y. Liu in *Scientific reports*, 2018.
16. *Latent ODEs for Irregularly-Sampled Time Series*, Y. Rubanova, R.T.Q. Chen, D. Duvenaud, *preprint*, 2019, arXiv:1907.03907.
17. *Neural Ordinary Differential Equations*, R.T.Q. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud in *NeurIPS*, 2018.
18. *Deep Learning with Sets and Point Clouds*, S. Ravanbakhsh, J. Schneider, B. Póczos, *preprint*, 2017, arXiv:1611.04500.
19. *On the Limitations of Representing Functions on Sets*, E. Wagstaff, F.B. Fuchs, M. Engelcke, I. Posner, M. Osborne, *preprint*, 2019, arXiv:1901.09006.
20. *Janossy Pooling: Learning Deep Permutation-Invariant Functions for Variable-Size Inputs*, R. Murphy, B. Srinivasan, V. Rao, B. Ribeiro, *preprint*, 2019, arXiv:1811.01900.
21. *Attention Is All You Need*, Vaswani, A., Shazeer, N., Parmer, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., and Polosukhin, I. in *NeurIPS*, 2017.
22. *Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks*, J. Lee, Y. Lee, J. Kim, A.R. Kosiorek, S. Choi, Y.W. Teh in *ICML*, 2019.

23. *Partially Exchangeable Networks and Architectures for Learning Summary Statistics in Approximate Bayesian Computation*, S. Wiqvist, P-A. Mattei, U. Picchini, J. Frellsen, *preprint*, 2019, arXiv:1901.10230.
24. *Neural Processes*, M. Garnelo, J. Schwarz, D. Rosenbaum, F. Viola, D.J. Rezende, S. M. Ali Eslami , Y.W. Teh, *preprint*, 2018, arXiv:1807.01622.
25. *Conditional Neural Processes*, M. Garnelo, D. Rosenbaum, C.J. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y.W. Teh, D.J. Rezende, S. M. Ali Eslami, *preprint*, 2018, arXiv:1807.01613.
26. *Probabilistic symmetries and invariant neural networks*, B. Bloem-Reddy, Y.W. Teh in *Journal of Machine Learning Research*, 2020.
27. *Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals.*, Goldberger, A. L., Amaral, L. A., Glass, L., Hausdorff, J. M., Ivanov, P., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C.-K., and Stanley, H. E. in *Circulation*, 2018.