Irregularly-Sampled Time Series Modeling with Spline Networks

Marin Biloš  Emanuel Ramneantu  Stephan Günnemann
Technical University of Munich, Germany

Abstract

Observations made in continuous time are often irregular and contain the missing values across different channels. One approach to handle the missing data is imputing it using splines, by fitting the piecewise polynomials to the observed values. We propose using the splines as an input to a neural network, in particular, applying the transformations on the interpolating function directly, instead of sampling the points on a grid. To do that, we design the layers that can operate on splines and which are analogous to their discrete counterparts. This allows us to represent the irregular sequence compactly and use this representation in the downstream tasks such as classification and forecasting. Our model offers competitive performance compared to the existing methods both in terms of the accuracy and computation efficiency.

1. Introduction

Irregularly-sampled data with missing values can be found in many domains. For example, medical records are a collection of multivariate attributes in continuous time, where only a small subset of possible measurements are taken at a certain time point. Both the missingness and the irregularity can be addressed by using some sort of interpolation on the raw data. One popular option is using splines, i.e., piecewise polynomials that fit the input points. The missing points can be imputed by simply evaluating the polynomial function at the target time points. Figure 1 demonstrates this: the original data is irregular and contains the missing values but after fitting the spline we can fill in the gaps. Additionally, we can evaluate the spline at any time point between the observed values to infer about the areas where we have not seen any data in the first place.

The existing methods such as RNNs, and neural ordinary and controlled differential equations use the imputed points but still need to take care of the irregular sampling. In their case, the latent state is evolved continuously over time. This approach is usually slower and the models are more likely to struggle with long-term dependencies, just like their discrete counterparts.

We propose an alternative where the imputed points are never materialized; rather, we work on the spline representation of the time series directly since it uniquely represents the data. That is, we use splines as an input to a neural network which in turn outputs a modified spline. We develop the layers that act as a continuous version of discrete layers, such as convolution and pooling operations. The final output of our model is the latent representation of the underlying time series, in the functional form. We can use this to obtain the unique fixed-sized latent vector that represents the whole time series, by simply querying the spline interpolation at the desired time points. The latent vector can be further used in the downstream tasks.

Correspondence to: Marin Biloš <marin.bilos@tum.de>.

Workshop on the Continuous Time Methods for Machine Learning, 39th International Conference on Machine Learning (ICML), Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).
A simple example is a linear spline that connects the knots \( x_i \) and \( x_{i+1} \) with straight lines. The coefficients of the linear function \( P_i(t) = a_1 t + a_0 \) can be easily calculated from data as \( a_1 = \frac{x_{i+1} - x_i}{t_{i+1} - t_i} \) and \( a_0 = x_i - a_1 t_i \). Besides linear, we can have piecewise constant (rectilinear) function that copies the previous value until the next observation, or higher order polynomials, cubic being the most popular one [8]. For cubic splines we require that the derivative in the knots matches the corresponding time points, i.e., that they are smooth. We get the coefficients by solving the system of equations that imposes this condition.

To evaluate the spline at an arbitrary time point \( t \), we first need to find the interval where \( t_i \leq t \leq t_{i+1} \) holds, then compute \( P_i(t) \). Consequently, evaluating \( S_x(t) \) at the observed time points \( (t_1, \ldots, t_n) \) performs the imputation of the missing values. At this point the previous methods [e.g., 17] would sample the points from the spline and use them in their models. We, instead, process the spline directly to obtain the useful \textit{continuous-in-time} features which, in turn, can be used to obtain, e.g., the fixed sized latent representation of the time series.

### 2.1. Spline operations

The operations on splines often reduce to operations on polynomials. For example, adding two splines that share the same set of time points \( (t_1, \ldots, t_n) \) results in adding the coefficients on the corresponding intervals. If the splines do not share the time points, we have to add the missing knots and define polynomials on newly generated intervals, as is shown in Figure 2a.

Another useful operation is the multiplication between two splines. This is again done piecewise. Multiplying two \( k \)th degree polynomials requires \( O(k^2) \) operations but can be reduced to \( O(k \log k) \) [see, e.g., 6, ch. 30]. The key insight is that the polynomial multiplication can be seen as a convolution. Thus, we can use the convolution theorem, i.e., apply the fast Fourier transform to the coefficients and perform the convolution as a pointwise product. This comes at a reduced computation cost compared to the naive approach. We get the resulting polynomial coefficients via inverse Fourier transform.

Now we tackle matching the knots of two arbitrary splines such that the result is two splines which share the knot positions exactly. If the spline has \( n \) knots and we want to add a new knot at time \( t \), we can simply find the interval \( [t_i, t_{i+1}] \) on which \( t \) falls and construct two new intervals \( [t_i, t] \) and \( [t, t_{i+1}] \). The first interval keeps the unchanged polynomial coefficients while the second one needs to shift the polynomial by \( t - t_i \). We shift the polynomials using the so-called Taylor shift algorithm [34] which has the complexity \( O(k \log k) \) for a \( k \)th degree polynomial. In general, when we have an operation
between two splines (e.g., a sum) with \( n \) and \( m \) knots, the worst case scenario gives us a spline with \( n + m \) knots.

### 2.2. SplineNet

Given a spline \( S_x \), we wish to transform the individual points using an affine layer \( S_y(t) = WS_x(t) + b \). The output function \( S_y \) will again be a spline with the same order as \( S_x \). This is easy to show since the affine function is multiplying the polynomials by a constant, adding them together, and finally, adding the constant to the result. Therefore, we can implement an affine layer by only handling the polynomial coefficients such that we never have to materialize the actual data points. Using such a layer allows for mixing between the time series channels (see Figure 2b). The resulting spline is defined with knots \( y \) that lie on the same time points as in \( x \), however, they have different values and dimension. This layer, in essence, projects the original interpolated time series to a continuous latent interpolation. It is a continuous version of the \( 1 \times 1 \) convolutional neural network.

An affine layer as defined above ignores the temporal dependencies since it processes points independently of their neighbors. Thus, we would like to have a way of aggregating the information over time. One approach is to perform the integration — a continuous equivalent of sum-pooling. That is, we wish to obtain \( \mathcal{I}_y(t) = \int_0^t S_y(\tau) d\tau \). Since \( S_y \) is a piecewise function, we can integrate over the individual sub-functions and add up the results. That means we need to calculate the integral of an arbitrary polynomial function \( P(t) = \sum_{i=0}^{k} c_i t^i \) which is simply \( \sum_{i=0}^{k+1} \frac{c_{i-1}}{i+1} t^i \) where \( c_{-1} \) is introduced as an offset that is calculated based on the value of the previous sub-functions. The resulting curve \( \mathcal{I}_y(t) \) is again a spline with an order \( k + 1 \).

Since integrating can lead to large values, we introduce a version of the batch normalization layer [13] that normalizes the area under the curve to have the mean equal to 1. The area statistics can be easily calculated by evaluating the integral spline at the last time point. Affine and integration layers act across the different channels and along the time dimension. For certain downstream tasks we want to represent the time series and its latent spline with a finite dimensional vector. One way to do this is to query the spline at the last observed time point \( t_a \). In this case, the proposed layers cannot distinguish certain splines, for example, integration aggregates the data without caring about the order in which it arrived.

To alleviate this, we add another type of layer where we multiply the input spline with a learnable spline kernel. The kernel is defined with learnable knot locations and polynomial coefficients. The result is again a spline. This corresponds to a convolution in the frequency domain. In practice, we use multiple kernels in parallel to extract more features. An alternative to multiplying the spline \( S_y \) with a spline kernel \( K \) is to use the kernel to learn the shape of the spline. That is, the output is now defined as the distance between the two splines: \( ||S_y - K||_2^2 \) (see Figure 2c). Integrating over this path gives us the distance between the two curves. The model essentially defines learnable shapes \( K \) which it tries to match with the input data shapes, giving us an interpretable layer, similar to the idea of shapelets [35].

Equipped with these layers, we can extract useful information from the original time series. However, if the time series is very long, the model has to learn the whole shape, but in practice, it might be better to extract the features locally. We can split the time series \( x \) into \( l \) smaller equally-spaced segments, apply the same layers as above and obtain \( l \) latent vectors. These vectors can then be passed to an RNN or some other established model for regular time series to get the unique latent representation of the whole input. Again, in practice, we do not actually need to split the time series; instead we query the final latent spline at \( l \) time points. The positions of these points can also be learnable.

### 2.3. Related work

The irregular data can be found in medicine [15], astronomy [28], climatology [29], and signal processing. Most of the approaches fill-in the missing values from the observed data either explicitly or implicitly, e.g., in non-uniform Fourier transform [10, 20], decision trees [7], linear regression [23], and neural networks [16, 32]. Existing methods that perform interpolation include gaussian processes [26], polynomial fitting, neural networks [30], or using piecewise functions. In our work we use splines since they are closed under our proposed model, i.e., the output of the neural network is again a spline.

Some recent approaches extend the RNNs to operate in continuous time [3, 24, 21]. Che et al. [3] introduce an extension of a GRU cell [5], that decays the hidden state between the observations. With the introduction of neural ODEs [4], several approaches proposed the evolution of the hidden state with a learnable ODE [9, 27, 14]. Since solving an ODE can be expensive, [2] introduce an alternative model that learns the solution curve directly. Kidger et al. [17] propose using neural controlled differential equations that use the continuous input to drive the latent representation using a reparameterized
Irregularly-Sampled Time Series Modeling with Spline Networks

|                          | Character (miss. 30%) | Character (miss. 70%) | Physionet | Speech Commands |
|--------------------------|-----------------------|-----------------------|-----------|-----------------|
| GRU                      | 0.9535 ± 0.012        | 0.9366 ± 0.025        | 0.9340 ± 0.006 | 0.8624 ± 0.001 |
| GRU-D                    | 0.9535 ± 0.023        | 0.9378 ± 0.016        | 0.9359 ± 0.006 | 0.8577 ± 0.012 |
| GRU-ODE / Flow           | 0.9634 ± 0.015        | 0.9476 ± 0.001        | 0.9353 ± 0.004 | 0.8426 ± 0.024 |
| Neural CDE               | 0.9407 ± 0.014        | 0.9546 ± 0.008        | 0.9352 ± 0.003 | 0.8608 ± 0.012 |
| Transformer              | 0.9785 ± 0.003        | 0.9715 ± 0.006        | 0.9181 ± 0.010 | 0.8325 ± 0.001 |
| SplineNet (Ours)         | **0.9849 ± 0.004**    | **0.9785 ± 0.003**    | **0.9400 ± 0.001** | **0.8693 ± 0.001** |

Table 1: Test accuracy mean and standard deviation, computed over 4 runs. For Physionet, we report AUC to account for data imbalance. Higher is better, the best performing model is in bold.

ODE. They use splines to construct the continuous path, which is evaluated multiple times during one solver call. Shukla & Marlin [31] extend transformers [33] by using the time attention and introduce the reference points on top of which RNN is used to output the final sequence representation. This is similar to our approach of dividing the time series into segments.

Shapelets [35, 12, 18] are interpretable feature extractors that aim to learn the shape of the input curve by sliding along the time series and computing the distance between the input points and the shapelet, which are used as an interpretable input to a downstream model. In our case, the shapes are learned splines that are fixed in position and potentially repeated across segments. We can also learn the shapes in the latent space which limits the interpretability but increases expressiveness. Since we avoid the expensive shapelet search algorithm, our method does not suffer from performance issues [25].

3. Experiments

In this section we demonstrate that our method can be used as an alternative to the established irregularly-sampled time series models through a set of different experiments such as classification, forecasting and probabilistic modeling. All the experiment were done on a single GPU with 12GB of memory.

Classification. We consider three datasets for our evaluation and process them similar to [17]. Character Trajectories [1] represents handwritten characters, each sequence has length 182 and 3 channels. To add irregularity, we create two augmented datasets with 30% and 70% of missing values, respectively. Physionet [11] is a medical dataset from hospitalized patients with the goal of predicting sepsis. It consists of 40,336 sequences with 34 time series features. There is a class imbalance, with 7% of the patients suffering sepsis and it contains many missing values (90% overall). Speech Commands dataset contains records with 35 classes corresponding to different spoken words, with sequences of length 161, each with 20 channels. All the datasets are normalized and we randomly split them into train, validation and test sets (60% – 20% – 20%).

Models: In order to assess the performance of our model we compared it to the GRU-D [3], two architectures based on differential equations, GRU-ODE [27] and Neural CDE [17], GRU Flow [2], and the transformer [31]. SplineNet model has affine, kernel convolution, integration, and ReLU layers as the building blocks. After applying these layers, we gather a single vector from each segment. This forms a fixed sized sequence which can then be fed to an RNN.

Results: In ablation studies, we notice that using a single block composing of an affine and integration layer followed by a convolution is enough to produce good results. Our experiments demonstrate that introducing integration makes by far the biggest difference, while the sizes of segments and shapes indicate that more is better in this case. Table 1 shows the results for all the baseline models and our model. Clearly, our model outperforms the others across all datasets. As for computation efficiency, our model is slower than transformers and much faster than ODE based models.

Probabilistic modeling. We define a latent variable model and set the experiment similar to [4, 27]. The comparison is between our encoder-decoder architecture based on SplineNet architecture and those using the neural ODEs. We report the reconstruction loss on the Character dataset with 70% missing values: GRU-ODE achieves 0.3924 ± 0.004, while our model outperforms it with 0.3088 ± 0.014.

Forecasting. We would like to encode the time series to predict the future values using the same encoder-decoder architectures as before. We use Physionet and Speech Commands dataset with the forecast window of 10 steps. Our method again outperforms the GRU-ODE model, we obtain reconstruction error of 0.0023 and 0.1736, compared to GRU’s 0.0025 and 0.1754, on Physionet and Speech, respectively.

Interpretability. As we already mentioned, the kernel layer with the distance measure can be used as an interpretable
layer since it learns the shapes of the input data, similarly to shapelets \cite{35}. Thus, we compare to \cite{18} who generalize the shapelets to the irregular data. We use the Japanese Vowels speech dataset with three different missingness levels and outperform the competitor in two out of three instances, failing to do so on the 50% missing data setting. The learned kernel splines that give the smallest distance to the data can be plotted to inspect which features contribute to the final prediction.

4. Discussion

In this paper we presented an alternative model for irregularly-sampled time series with missing data. Our method outperforms the competitors, such as the ODE-based models and transformers, on a variety of tasks. We also offer the ability to inspect the inner workings of our layers, which can be interpretable as they learn the shapes in the data. One limitation of our work is the memory footprint which may be larger than in the other models depending on the number of learnable kernels. We did not, however, face any issues in our experiments because of this.

In the future, we can utilize more of the spline properties. For example, we could use regularized splines that do not pass through all the data points to drop the outliers. Further, we can have segments of varying sizes, acting on the time series simultaneously to obtain the hierarchical features, allowing learning at different scales.

References

\[1\] Bagnall, A., Dau, H. A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., and Keogh, E. The UEA multivariate time series classification archive, 2018. \textit{arXiv preprint arXiv:1811.00075}, 2018.

\[2\] Biloš, M., Sommer, J., Rangapuram, S. S., Januschowski, T., and Günnemann, S. Neural flows: Efficient alternative to neural ODEs. In \textit{NeurIPS}, 2021.

\[3\] Che, Z., Purushotham, S., Cho, K., Sontag, D., and Liu, Y. Recurrent neural networks for multivariate time series with missing values. \textit{Scientific reports}, 8(1):1–12, 2018.

\[4\] Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In \textit{NeurIPS}, 2018.

\[5\] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. In \textit{Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8)}, 2014.

\[6\] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. \textit{Introduction to algorithms}. MIT press, 2009.

\[7\] Cutler, A., Cutler, D. R., and Stevens, J. R. Random forests. In \textit{Ensemble machine learning}, pp. 157–175. Springer, 2012.

\[8\] De Boor, C. \textit{A practical guide to splines}, volume 27. Springer-Verlag New York, 1978.

\[9\] De Brouwer, E., Simm, J., Arany, A., and Moreau, Y. GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series. In \textit{NeurIPS}, 2019.

\[10\] Dutt, A. and Rokhlin, V. Fast fourier transforms for nonequispaced data. \textit{SIAM Journal on Scientific computing}, 14 (6):1368–1393, 1993.

\[11\] Goldberger, A. L., Amaral, L. A. N., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C.-K., and Stanley, H. E. Physiobank, PhysioToolkit, and Physionet: Components of a new research resource for complex physiologic signals. \textit{Circulation}, 101(23):e215–e220, 2000.

\[12\] Grabocka, J., Schilling, N., Wistuba, M., and Schmidt-Thieme, L. Learning time-series shapelets. In \textit{KDD}, 2014.

\[13\] Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In \textit{International conference on machine learning}, pp. 448–456. PMLR, 2015.

\[14\] Jia, J. and Benson, A. R. Neural jump stochastic differential equations. In \textit{NeurIPS}, 2019.
Irregularly-Sampled Time Series Modeling with Spline Networks

[15] Johnson, A., Bulgarelli, L., Pollard, T., Horng, S., Celi, L. A., and Mark, R. MIMIC-IV (version 1.0). PhysioNet, 2021. doi: 10.13026/s6n6-xd98.

[16] Khosravi, P., Choi, Y., Liang, Y., Vergari, A., and Broeck, G. V. d. On tractable computation of expected predictions. In NeurIPS, 2019.

[17] Kidger, P., Morrill, J., Foster, J., and Lyons, T. J. Neural controlled differential equations for irregular time series. In NeurIPS, 2020.

[18] Kidger, P., Morrill, J., and Lyons, T. Generalised interpretable shapelets for irregular time series. arXiv preprint arXiv:2005.13948, 2020.

[19] Lechner, M. and Hasani, R. Learning long-term dependencies in irregularly-sampled time series. In NeurIPS, 2020.

[20] Lomb, N. R. Least-squares frequency analysis of unequally spaced data. Astrophysics and space science, 39(2):447–462, 1976.

[21] Mei, H. and Eisner, J. M. The neural hawkes process: A neurally self-modulating multivariate point process. In NeurIPS, 2017.

[22] Moritz, S. and Bartz-Beielstein, T. imputets: time series missing value imputation in r. R J., 9(1):207, 2017.

[23] Morvan, M. L., Josse, J., Moreau, T., Scornet, E., and Varoquaux, G. Neumiss networks: differentiable programming for supervised learning with missing values. In NeurIPS, 2020.

[24] Neil, D., Pfeiffer, M., and Liu, S.-C. Phased LSTM: Accelerating recurrent network training for long or event-based sequences. In NeurIPS, 2016.

[25] Rakthanmanon, T. and Keogh, E. Fast shapelets: A scalable algorithm for discovering time series shapelets. In Proceedings of the 2013 SIAM International Conference on Data Mining. SIAM, 2013.

[26] Rasmussen, C. E. and Williams, C. K. I. Gaussian processes for machine learning. MIT Press, 2006.

[27] Rubanova, Y., Chen, R. T., and Duvenaud, D. Latent ODEs for irregularly-sampled time series. In NeurIPS, 2019.

[28] Scargle, J. D. Studies in astronomical time series analysis. ii-statistical aspects of spectral analysis of unevenly spaced data. The Astrophysical Journal, 263:835–853, 1982.

[29] Schulz, M. and Stattegger, K. Spectrum: Spectral analysis of unevenly spaced paleoclimatic time series. Computers & Geosciences, 23(9):929–945, 1997.

[30] Shukla, S. N. and Marlin, B. M. Interpolation-prediction networks for irregularly sampled time series. In ICLR, 2019.

[31] Shukla, S. N. and Marlin, B. M. Multi-time attention networks for irregularly sampled time series. In ICLR, 2021.

[32] Smieja, M., Struski, Ł., Tabor, J., Zieliński, B., and Spurek, P. Processing of missing data by neural networks. In NeurIPS, 2018.

[33] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In NeurIPS, 2017.

[34] Von Zur Gathen, J. and Gerhard, J. Fast algorithms for taylor shifts and certain difference equations. In Proceedings of the 1997 international symposium on Symbolic and algebraic computation, pp. 40–47, 1997.

[35] Ye, L. and Keogh, E. Time series shapelets: a new primitive for data mining. In KDD, 2009.