Missing Value Imputation on Multidimensional Time Series

Parikshit Bansal
IIT Bombay
parikshitb52@gmail.com

Prathamesh Deshpande
IIT Bombay
pratham@cse.iitb.ac.in

Sunita Sarawagi
IIT Bombay
sunita@iitb.ac.in

ABSTRACT

We present DeepMVI, a deep learning method for missing value imputation in multidimensional time-series datasets. Missing values are commonplace in decision support platforms that aggregate data over long time stretches from disparate sources, whereas reliable data analytics calls for careful handling of missing data. One strategy is imputing the missing values, and a wide variety of algorithms exist spanning simple interpolation, matrix factorization methods like SVD, statistical models like Kalman filters, and recent deep learning methods. We show that often these provide worse results on aggregate analytics compared to just excluding the missing data.

DeepMVI expresses the distribution of each missing value conditioned on coarse and fine-grained signals along a time series, and signals from correlated series at the same time. Instead of resorting to linearity assumptions of conventional matrix factorization methods, DeepMVI harnesses a flexible deep network to extract and combine these signals in an end-to-end manner. To prevent over-fitting with high-capacity neural networks, we design a robust parameter training with labeled data created using synthetic missing blocks around available indices. Our neural network uses a modular design with a novel temporal transformer with convolutional features, and kernel regression with learned embeddings.

Experiments across ten real datasets, five different missing scenarios, comparing seven conventional and three deep learning methods show that DeepMVI is significantly more accurate, reducing error by more than 50% in more than half the cases, compared to the best existing method. Although slower than simpler matrix factorization methods, we justify the increased time overheads by showing that DeepMVI provides significantly more accurate imputation that finally impacts quality of downstream analytics.

PVLDB Reference Format:
Parikshit Bansal, Prathamesh Deshpande, and Sunita Sarawagi. Missing Value Imputation on Multidimensional Time Series. PVLDB, 14(11): 2533-2545, 2021.
doi:10.14778/3476249.3476300

PVLDB Availability Tag:
The source code of this research paper has been made publicly available at https://github.com/pbansal5/DeepMVI.

1 INTRODUCTION

In this paper we present a system for imputing missing values across multiple time series occurring in multidimensional databases. Examples of such data include sensor recordings along time of different types of IoT devices at different locations, daily traffic logs of web pages from various device types and regions, and demand along time for products at different stores. Missing values are commonplace in analytical systems that integrate data from multiple sources over long periods of time. Data may be missing because of errors or breakdowns at various stages of the data collection pipeline ranging from faulty recording devices to deliberate obfuscation. Analysis on such incomplete data may yield biased results misinforming data interpretation and downstream decision making. Therefore, missing value imputation is an essential tool in any analytical systems [3, 10, 18, 21].

Many techniques exist for imputing missing values in time-series datasets including several matrix factorization techniques [2, 11, 19, 20, 24, 28], statistical temporal models [14], and recent deep learning methods [4, 8]. Unfortunately, even the best of existing techniques still incur high imputation errors. We show that top-level aggregates used in analytics could get worse after imputation with existing methods, compared to discarding missing data parts before aggregation. Inspired by the recent success of deep learning in other data analytical tasks like entity matching, entity extraction, and time series forecasting, we investigate if better deep learning architectures can reduce this gap for the missing value imputation task.

The pattern of missing blocks in a time series dataset can be quite arbitrary and varied. Also, datasets could exhibit very different characteristics in terms of the length and number of series, amount of repetitions (seasonality) in a series, and correlations across series. An entire contiguous block of entries might be missing within a time series, and/or across multiple time-series. The signals from the rest of the dataset that are most useful for imputing a missing block would depend on the size of the block, its position relative to other missing blocks, patterns within a series, and correlation (if any) with other series in the dataset. If a single entry is missing, interpolation with immediate neighbors might be useful. If a range of values within a single time series is missing, repeated patterns within the series and trends from correlated series might be useful. If the same time range across several series is missing, only patterns within a series will be useful.

Existing methods based on matrix factorization can exploit across series correlations but are not as effective in combining them with temporal patterns within a series. Modern deep learning methods, because of their higher capacity and flexibility, can in principle combine diverse signals when trained end to end. However, designing a neural architecture whose parameters can be trained accurately and scalably across diverse datasets and missing patterns proved to be non-trivial. Early solutions based on popular architectures for sequence data, such as recurrent neural networks (RNNs) have been shown to be worse both in terms of accuracy and running time. We explored a number of alternative architectures spanning

Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476300

Transformers, CNNs, and Kernel methods. A challenge we faced when training a network to combine a disparate set of potentially useful signals was that, the network was quick to overfit on easy signals. Whereas robust imputation requires that the network harness all available signals. After several iterations, we converged on a model and training procedure, that we call DeepMVI that is particularly suited to the missing value imputation task.

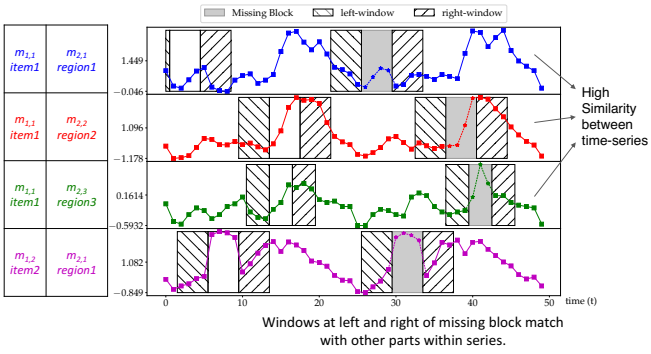


Figure 1: Grey-shaded regions denote missing blocks. Patterns of left and right windows around each missing block match with another part of the same series. Series 1-3 have high similarity and series 1,2,4 show good window match along time.

1.1 Contributions

(1) We propose a tractable model to express each missing value using a distribution conditioned on available values at other times within the series and values of similar series at the same time. (2) We design a flexible and modular neural architecture to extract fine-grained, coarse-grained, and cross-series signals to parameterize this distribution. (3) We provide a robust method of training generalizable parameters by simulating missing patterns around available indices that are identically distributed to the actual missing entries. (4) Our neural architecture includes a Temporal transformer that differs from off-the-shelf Transformers in our method of creating contextual keys used for self-attention. (5) We propose the use of Kernel regression for incorporating information from correlated time-series. This method of extracting relatedness is scalable and extends naturally to multidimensional datasets, that none of the existing methods handle. (6) We achieve 20–70% reduction in imputation error as shown via an extensive comparison with both state of the art neural approaches and traditional approaches across ten real-life datasets and five different patterns of missing values. We show that this also translates to more accurate aggregates analytics on datasets with missing entries. (7) Our method is six times faster than using off-the-shelf deep-learning components for MVI.

2 PRELIMINARIES AND RELATED WORK

We present a formal problem statement, discuss related work, and provide background on relevant neural sequence models.

2.1 Problem Statement

We denote our multidimensional time series dataset as an $n+1$ dimensional data tensor of real values $X \in \mathbb{R}^{n+1}$. The dimensions of X are denoted as (K_1,K_2,...,K_n, K_{n+1}). The dimension K_{n+1} denotes a regularly spaced time index which, without loss of generality we denote as $\{1, \ldots, T\}$. Each K_i is a dimension comprising of a discrete set of members $\{m_{i,1}, \ldots, m_{i,|K_i|}\}$. Each member m_{ij} could be either a categorical string or a real-valued vector. For example, a retail sales data might consist of two such dimensions: K_1 comprising of categorical members denoting identity of items sold and K_2 comprising of stores where a store is defined in terms of its continuous latitude and longitude value. We denote a specific combination of members of each dimension as $\mathbf{k} = k_1, \ldots, k_n$ where each $k_i \in \text{Dim}(K_i)$. We refer to the value at a combination $\mathbf{k}$ and time t as $X_{\mathbf{k},t}$. For example in Figure 1 we show four series of length 50 and their index $\mathbf{k}$ sampled from a two dimensional categorical space of item-ids and region-ids. We are given an X with some fraction of values $X_{\mathbf{k},t}$ missing. Let M and A be tensors of only ones and zeros with same shape as X that denote the missing and available values respectively in X. We use $\mathcal{I}(M)$ to denote all missing values' $(\mathbf{k}, t)$ indices. The patterns in $\mathcal{I}(M)$ of missing index combinations can be quite varied — for example missing values may be in contiguous blocks or isolated points; across time-series the missing time-ranges may be overlapping or missing at random; or in an extreme case called Blackout a time range may be missing in all series. Our goal is to design a procedure that can impute the missing values at the given indices $I(M)$ so that the error between the imputed values $\hat{X}$ and ground truth values $\bar{X}$ is minimized.

$$\sum_{(\mathbf{k},t) \in \mathcal{I}(M)} \mathcal{E}(\hat{X}_{\mathbf{k},t}, \bar{X}_{\mathbf{k},t}) \tag{1}$$

where $\mathcal{E}$ denotes error functions such as root mean square error (RMSE) and mean absolute error (MAE). As motivated in Figure 1 both patterns within and across a time series may be required to fill a missing block.

2.2 Related Work

Missing value imputation in time series is an age-old problem [16], with several solutions that we categorize into matrix-completion methods, conventional statistical time-series models, and recent deep learning methods (discussed in Section 2.4). However, all these prior methods are for single-dimensional series. So, we will assume $n = 1$ for the discussions below.

Matrix completion methods These methods [2, 11, 19, 20, 24, 28], view the time-series dataset as a matrix X with rows corresponding to series and columns corresponding to time. They then apply various dimensional reduction techniques to decompose the matrix as $X \approx UV^T$ where U and V represent low-dimensional embeddings of series and time respectively. The missing entry in a series i and position t is obtained by multiplying the corresponding embeddings. A common tool is the classical Singular Value Decomposition (SVD) and this forms the basis of three earlier techniques: SVDImp[24], SoftImpute [19], and SVT [2]. All these methods are surpassed by a recently proposed centroid decomposition (CD) algorithm called CDRec[11]. CDRec performs recovery by first using interpolation/extrapolation to initialize the missing values. Second,

it computes the CD and keeps only the first k columns of U and V, producing U_k and V_k, respectively. Lastly, it imputes values using $X = U_k V_k^T$. This process iterates until the normalized Frobenius norm between the matrices before and after the update reaches a small threshold.

A limitation of pure matrix decomposition based methods is that they do not capture any dependencies along time. TRMF[28] proposes to address this limitation by introducing a regularization on the temporal embeddings V so that these confirm to auto-regressive structures commonly observed in time-series data. STMVL is another algorithm that smooths along time and is designed to recover missing values in spatio-temporal data using collaborative filtering methods for matrix completion.

Statistical time-series models DynaMMO[14], is an algorithm that creates groups of a few time series based on similarities that capture co-evolving patterns. They fit a Kalman Filter model on the group using the Expectation Maximization (EM) algorithm. The Kalman Filter uses the data that contains missing blocks together with a reference time series to estimate the current state of the missing blocks. The recovery is performed as a multi-step process. At each step, the EM method predicts the value of the current state and then two estimators refine the predicted values of the given state, maximizing a likelihood function.

Pattern Based Methods TKCM [26] identifies and uses repeating patterns (seasonality) in the time series' history. They find similarity between window of measures spanning all time series and window around the query time index using Pearson's correlation coefficient and do 1-1 imputation using the mean value of the matched blocks. Though promising this method performs poorly compared to other baselines like CDRec, on each dataset [12], hence we have excluded it from our analysis. Deep Learning architectures have been shown to perform better at query-pattern search and corresponding weighted imputation [25] which we exploit in our work. We present an empirical comparison with SVDImp (as a representative of pure SVD methods), CDRec, TRMF, STMVL, and DynaMMO and show that our method is significantly more accurate than all of them.

2.3 Background on Neural Sequence Models

We review[1] two popular neural architectures for processing sequence data.

2.3.1 Bidirectional Recurrent Neural Networks. Bidirectional RNN [9] is a special type of RNN that captures dependencies in a sequence in both forward and backward directions. Unlike forecasting, context in both forward and backward directions is available in MVI task. Bidirectional RNN maintains two sets of parameters, one for forward and another for backward direction. Given a sequence X, the forward RNN maintains a state $\mathbf{h}_t^f$ summarizing $X_1 \dots X_{t-1}$, and backward RNN maintains a state $\mathbf{h}_t^b$ summarizing $X_T \dots X_{t+1}$. These two states jointly can be used to predict a missing value at t. Because each RNN models the dependency along only one direction, a bidirectional RNN can compute loss at each term in the input during training.

[1]Readers familiar with Deep Learning may skip this subsection.

2.3.2 Transformers. A Transformer [25] is a special type of feed-forward neural network that captures sequential dependencies through a combination of self-attention and feed-forward layers. Transformers are primarily used on text data for language modelling and various other NLP tasks [6], but have recently also been used for time-series forecasting [15].

Given an input sequence X of length T, a transformer processes it as follows: It first embeds the input X_t for each $t \in [1, T]$ into a vector $E_t \in \mathbb{R}^p$, called the input embedding. It also creates a positional encoding vector at position t denoted as $e_t \in \mathbb{R}^p$.

$$e_{t,r} = \begin{cases} \sin(t/10000^{\frac{r}{p}}), & \text{if } r\%2 == 0 \\ \cos(t/10000^{\frac{r-1}{p}}), & \text{if } (r-1)\%2 == 0 \end{cases} \quad (2)$$

Then it uses linear transformation of input embedding and positional encoding vector to create query, key, and value vectors.

$$Q_t = (E_t + e_t)W^Q \quad K_t = (E_t + e_t)W^K \quad V_t = (E_t + e_t)W^V \quad (3)$$

where the Ws denote trained parameters. Key and Value vectors at all times $t \in [1, T]$ are stacked to create matrices K and V respectively. Then the query vector at time t and keys pair at other positions $t' \neq t$ are used to compute a self-attention distribution, which is used to compute a vector at each t as an attention weighted sum of its neighbors as follows:

$$\mathbf{h}_t = \text{Softmax}(\frac{Q_t K^T}{\sqrt{p}})V \quad (4)$$

Such self-attention can capture the dependencies between various positions of the input sequence. Transformers use multiple such self-attentions to capture different kinds of dependencies, and these are jointly referred as multi-headed attention. In general, multiple such layers of self-attention are stacked. The final vector $\mathbf{h}_t$ at each t presents a contextual representation of each t.

For training the parameters of the transformer, a portion of the input would be masked (replaced by 0). We denote the masked indices by M. The training loss is computed only on the masked indices in M. This is because multiple layers of self-attention can compute $\mathbf{h}_t$ as a function of any of the input values. This is unlike bidirectional RNNs where the forward and backward RNN states clearly demarcate the values used in each state. This allow loss to be computed at each input position. However, transformers are otherwise faster to train in parallel unlike RNNs.

2.4 Related work in Deep-learning

In spite of the recent popularity and success of deep learning (DL) in several difficult tasks, existing work on the MVI task are few in number. Also there is limited evidence of DL methods surpassing conventional methods across the board. MRNN[27] is one of the earliest deep learning methods. MRNNs use Bidirectional RNNs to capture context of a missing block within a series, and capture correlations across series using a fully connected network. However, a detailed empirical evaluation in [12] found MRNN to be orders of magnitude slower than above matrix completion methods, and also (surprisingly) much worse in accuracy. More recently, BRITS[4] is another method that also uses bidirectional RNNs. At each time step t the RNN is input a column $X_{:,t}$ of X. The RNN state is the black box charged with capturing both the dependencies across time and across series. GP-VAE[8] adds more structure to

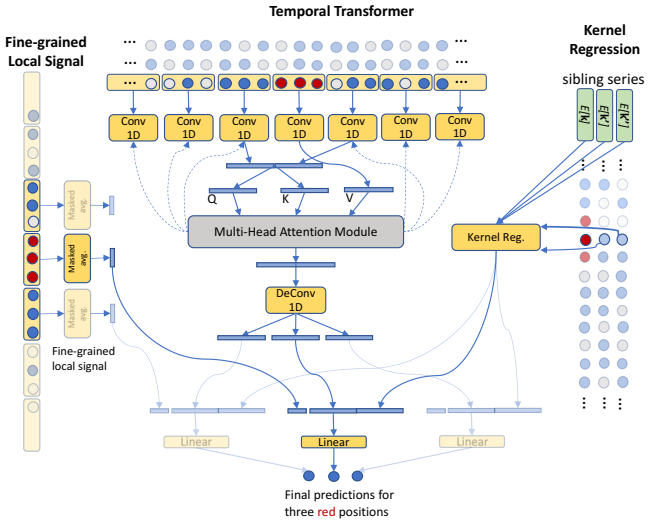


Figure 2: Architecture of DeepMVI. Here model is shown imputing the three circles marked in red at the top. The temporal transformer convolves on window of size $w = 3$ to create queries, keys and values for the multi-headed attention. The deconvolution creates three vectors, one for each red circle. These are concatenated with fine-grained signal and kernel regression to predict the final output.

the dependency by first converting each data column $X_{:,t}$ of X to a low-dimensional embedding, and then using a Gaussian Process to capture dependency along time in the embedding space. Training is via an elaborate structured variational method. On record time series datasets GP-VAE has been shown to be worse empirically than BRITS, and seems to be geared towards image datasets.

Compared to these deep models, our network architecture is more modular and light-weight in design, allows for more stable training without dataset specific hyper-parameter tuning, more accurate, and significantly faster.

Other Deep Temporal Models Much of the work on modeling time series data has been in the context of the forecasting task. State of the art methods for forecasting are still RNN-based [5, 7, 22, 23]. The only exceptions is [15] that uses convolution to extract local context features of the time-series and then a transformer to capture longer-range features. Such transformer and convolutions models have been quite successful in speech transcription literature [13]. Our architecture is also based on transformers and convolutions but our design of the keys and queries is better suited for missing value imputation. Further, we also include a fine-grained context and a second kernel regression model to handle across time correlations.

3 DEEPMVI: THE CONCEPTUAL MODEL

We cast the missing value imputation task as solving an objective of the form:

$$\max_{\hat{X}} \prod_{(\mathbf{k},t)\in\mathcal{I}(M)} \Pr(\hat{X}_{\mathbf{k},t}|X,A;\theta)$$

where θ are the parameters of the model and A is the mask denoting available values in X. X is the entire set of available values, and any tractable models will need to break-down the influence of X at an $(\mathbf{k},t)$ into simpler, learnable subparts. State of the art deep learning methods such as BRITS, simplify the dependence as:

$$\Pr(\hat{X}_{\mathbf{k},t}|X,A;\theta) = \Pr(\hat{X}_{\mathbf{k},t}|X_{\bullet,1\ldots t-1}, X_{\bullet,t+1\ldots T},\theta)$$

The first part $X_{\bullet,1\ldots t-1}$ denotes the entire vector of values over all times before t and likewise $X_{\bullet,t+1\ldots T}$ for after t. Note here observed values at time t from correlated sequences are ignored. Also, each of these sequences are summarized using RNNs that take as input values over all series $X_{\bullet,j}$ at each step j. This limits the scalability on the number of series.

In contrast, DeepMVI captures the dependence both along time within series $\mathbf{k}$ and along other related series at time t to simplify the dependency structure as:

$$\Pr(\hat{X}_{\mathbf{k},t}|X,A;\theta) = \Pr(\hat{X}_{\mathbf{k},t}|X_{\mathbf{k},\bullet}, X_{\mathrm{Sib}(\mathbf{k}),t},A,\theta) \quad (5)$$

The first part $X_{\mathbf{k},\bullet}$ is used to capture the long term dependency within the series $\mathbf{k}$ and also fine-grained signals from the immediate temporal neighborhood of t. The second part extracts signals from related series $\mathrm{Sib}(\mathbf{k})$ at time t. The notion of $\mathrm{Sib}(\mathbf{k})$ is defined using learnable kernels that we discuss in Section 4.2. Unlike conventional matrix factorization or statistical methods like Kalman Filters that assume fixed functional forms, we depend on the universal approximation power of neural networks to extract signals from the context $X_{\mathbf{k},\bullet}$ and $X_{\mathrm{Sib}(\mathbf{k}),t}$ to create a distribution over the missing value at $\mathbf{k},t$. The neural network architecture we used for parameterizing this distribution is described Section **??**.

The parameters (θ) of high-capacity neural networks need to be trained carefully to avoid over-fitting. We do not have separate labeled datasets for training the model parameters. Instead, we need to create our own labeled dataset using available values A in the same data matrix X.

We create a labeled dataset from randomly sampled $(\mathbf{k}_i,t_i)$ indices from the available set A. Each index $(\mathbf{k}_i,t_i)$ defines a training instance with input $\mathbf{x}_i = (X_{\mathbf{k},\bullet}, X_{Sib(\mathbf{k}),t}, A)$ and continuous output $y_i = X_{\mathbf{k}_i,t_i}$ as label, and thus can be cast as a standard regression model. In order for the trained parameters θ to generalize to the missing indices in $\mathcal{I}(M)$, the available values in the context of a $(\mathbf{k}_i,t_i)$ used in training need to be distributed identically to those in $\mathcal{I}(M)$. We achieve this by creating synthetic missing values around each $(\mathbf{k}_i,t_i)$. The shape of the missing block is chosen by sampling a shape B_i from anywhere in M. Note the shape B_i is a cuboid characterized by just the *number* (and not the position) of missing values along each of the $n+1$ dimensions. We then place B_i randomly around $(\mathbf{k}_i,t_i)$, to create a new availability matrix A_i after masking out the newly created missing entries around $(\mathbf{k}_i,t_i)$. Our training objective thereafter is simple likelihood maximization over the training instances as follows:

$$\theta^* = \mathrm{argmax}_\theta \sum_{(\mathbf{k}_i,t_i)\in\mathcal{I}(A)} [\log \Pr(X_{\mathbf{k},t}|X_{\mathbf{k},\bullet}, X_{\mathrm{Sib}(\mathbf{k}),t}, A^i,\theta)]$$

Thus θ^* has been trained to predict the true value of $|A|$ instances, where our method of sampling A^i ensures that these are identically distributed as the missing entries $\mathcal{I}(M)$. We further prevent overfitting on the training instances by using a validation dataset to do

early stopping. This implies that we can invoke the standard ML guarantees of generalization to claim that our model will generalize well to the unseen indices.

4 DEEPMVI: THE NEURAL ARCHITECTURE

We implement the conditional model of Equation 5 as a multi-layered modular neural network. The first module is the temporal transformer that takes as input $X_{\mathbf{k},\bullet}$ and extracts two types of imputation signals along time: longer-term seasonality represented as a output vector $\mathbf{h}^{\text{tt}} = TT_\theta(X_{\mathbf{k},\bullet}, A_{\mathbf{k},\bullet})$, and a fine-grained signal from the immediate neighborhood of $(\mathbf{k}, t)$ represented as $\mathbf{h}^{\text{fg}} = FG_\theta(X_{\mathbf{k},\bullet}, A_{\mathbf{k},\bullet})$. The second module is the kernel regression that extracts information from related series at time t i.e., from $X_{\text{Sib}(\mathbf{k}),t}$ to output another hidden vector $\mathbf{h}^{\text{kr}} = KR_\theta(X_{\bullet,t}, A\bullet, t)$. The last layer combines these three outputs as a light-weight linear layer to output a mean value of the distribution of $\hat{X}_{\mathbf{k},t}$ as follows:

$$\mu[\hat{X}_{\mathbf{k},t}] = \boldsymbol{w}_o^T [\mathbf{h}^{\text{tt}}, \mathbf{h}^{\text{fg}}, \mathbf{h}^{\text{kr}}] + \boldsymbol{b}_o \tag{6}$$

The above mean is used to model a Gaussian distribution for the conditional probability with a shared variance. A pictorial depiction of our pipeline appears in Figure 2. We describe each of these modules in the following sections.

4.1 Temporal Transformer Module

We build this module for capturing temporal dependency in the series akin to seasonality and draw inspiration from the Transformer [25] architecture. The parallel processing framework of attention module provides a natural formulation for handling missing values by masking inputs in contrast to the sequential modeling in RNNs. However, our initial attempts at using the vanilla Transformer model (described in Section 2.3.2) for the MVI task was subject to over-fitting on long time-series, and inaccurate for block missing values. We designed a new transformer specifically suited for the MVI task which we call the Temporal Transformer.

With the slight abuse of notation we override the definition of data and availability tensors X and A to 1-dimensional data and availability series respectively. Accordingly $I(A)$ and $I(M)$ are also overridden to confirm to series data. We use $I(A) = I - I(M)$ to denote all the indices that are not missing in X. We next describe how Temporal Transformer computes the function $TT_\theta(X, A)$.

Window-based Feature Extraction A linear operation on the window $X_{jw:(j+1)w}$ computes a p-dimensional vector Y_j as follows:

$$Y_j = W_f X_{jw:(j+1)w} + b_f \tag{7}$$

where $W_f \in \mathbb{R}^{p \times w}$ and $b_f \in \mathbb{R}^p$ are parameters. This operation is also termed as non-overlapping convolutions in Deep Learning literature.

We use self-attention on Y_j vectors obtained from Eqn. 7 above. We now describe the computation of query, key and value vectors for the self-attention.

Query, Key and Value functions For an index j (corresponding to the vector Y_j), we define the functions query and key functions $Q(\cdot)$ and $K(\cdot)$ as the functions of Y_{j-1} and Y_{j+1} respectively. Similarly we define value $V(\cdot)$ as function of Y_j.

$$Q(Y, j) = ([Y_{j-1}, Y_{j+1}] + e_j)W_q + b_q \tag{8}$$

$$K(Y, j, A) = (([Y_{j-1}, Y_{j+1}] + e_j)W_k + b_k) \cdot \prod_{i=jw}^{(j+1)w} A_i \tag{9}$$

$$V(Y_j) = Y_j W_v + b_v \tag{10}$$

where $W_q, W_k \in \mathbb{R}^{2p \times 2p}$, $W_v \in \mathbb{R}^{p \times p}$, e_j is the positional encoding of index j defined in Eqn. 2. The product of A_i-s in Eqn. 9 is one only if all values in the window are available. This prevents attention on windows with missing values. Note that keys and values are calculated for other indices $j' \neq j$ as well.

Attention Module Attention module computes the attention-weighted sum of vectors $V(Y_\bullet)$. The attention-weighted sum at index j is calculated as follows:

$$\text{Attn}(Q(\cdot), K(\cdot), V(\cdot), A, j) = \frac{\sum_{j'} \langle Q(Y, j), K(Y, j', A) \rangle V(Y_{j'})}{\sum_{j'} \langle Q(Y, j), K(Y, j', A) \rangle} \tag{11}$$

Note that for indices j' with missing values, including the index j, the key $K(Y, j', A)$ is zero. Hence such indices are not considered in the attention.

MultiHead Attention Instead of using only one attention module, our temporal transformer uses multiple instantiations of the functions $Q(Y, j), K(Y, j, A), V(Y_j)$. We compute n_{head} such instantiations and denote them using index $l = 1 \ldots n_{\text{head}}$.

We obtain the output of multi-head attention by concatenating the output vectors of all n_{head} attentions (obtained from Eqn. 11) into a vector $\mathbf{h} \in \mathbb{R}^{p n_{\text{head}}}$.

$$\mathbf{h}_j = [\text{Attn}^1(\cdots), \ldots, \text{Attn}^{n_{\text{head}}}(\cdots)] \tag{12}$$

Decoding Attention Output Vector $\mathbf{h}_j$ is the output vector for window $X_{jw:(j+1)w}$. Decoding module first passes the vector $\mathbf{h}_j$ through a feed-forward network to obtain the vector $\mathbf{h}_j^{\text{ff}}$. It then transforms this vector to obtain the output vectors for positions $\{jw, \ldots, t, \ldots, (j+1)w\}$:

$$\mathbf{h}_j^{\text{ff}} = \text{ReLU}(W_{d_2}(\text{ReLU}(W_{d_1}(\text{ReLU}(\mathbf{h}_j))))) \tag{13}$$

$$\mathbf{h}_j^{\text{tt}} = \text{ReLU}(W_d \mathbf{h}_j^{\text{ff}} + b_d) \tag{14}$$

where $W_d \in \mathbb{R}^{w \times p \times p}$. Note that $\mathbf{h}_j^{\text{tt}} \in \mathbb{R}^{w \times \bullet}$ consists of output vectors for all indices in the j-th window. We can obtain the output vector corresponding to index t as $\mathbf{h}^{\text{tt}} = \mathbf{h}_j^{\text{tt}}[t\%w]$ as the final output of the $TT_\theta(X, A, t)$ module.

4.1.1 Fine Grained Attention Module. We utilise this module to capture the local structure from immediate neighbouring time indices which are especially significant in the case of point missing values. Let the time index t be part of the window $j = \lfloor \frac{t}{w} \rfloor$ with start and end times as t_s^j and t_e^j respectively. Then we define the function $FG_\theta(X, A)$ as

$$FG_\theta(X, A) = \mathbf{h}^{\text{fg}} = \frac{\sum_{j \in I(A)} X_j}{|I(A)|} \tag{15}$$

4.2 Kernel Regression Module

We build the kernel regression module to exploit information from correlated series along each of the n data dimensions. A series in our case is associated with n variables $\mathbf{k} = k_1, \ldots, k_n$.

Index Embeddings First, we embed each dimension member in a space that preserves their relatedness. If a member m_{ij} of a dimension K_i is categorical we learn an embedding vector $E_\theta(m_{ij}) \in R^{d_i}$. When the dimension is real-valued, i.e., $m_{ij} \in R^p$, we use $E_\theta(m_{ij})$ to denote any feed-forward neural network to transform the raw vector into a d_i-dimensional vector.

We define relatedness among series pairs. We only consider series pairs that differ in exactly one dimension. We call these sibling series:

Defining Siblings We define siblings for an index $\mathbf{k}$ along dimension i, $\text{Sib}(\mathbf{k}, i)$ as the set of all indices $\mathbf{k}'$ such that $\mathbf{k}$ and $\mathbf{k}'$ differ only at i-th dimension.

$$\text{Sib}(\mathbf{k}, i) = \{\mathbf{k}' : k'_j = k_j \; \forall j \neq i \wedge k'_i \neq k_i\} \tag{16}$$

Here we override the notation $\text{Sib}(\mathbf{k})$ (used earlier in Eqn. 5) to identify the siblings along each dimension. For example, in a retail sales data, that contains three items $\{i_0, i_1, i_2\}$ and four regions $\{r_0, r_1, r_2, r_3\}$, siblings of an (item, region) pair $\mathbf{k} = (i_1, r_2)$ along the product dimension would be $\text{Sib}(\mathbf{k}, 0) = \{(i_0, r_2), (i_2, r_2)\}$ and along the region dimension would be $\text{Sib}(\mathbf{k}, 1) = \{(i_1, r_0), (i_1, r_1), (i_1, r_3)\}$.

Regression along each dimension An RBF Kernel computes the similarity score $\mathcal{K}(k_i, k'_i)$ between indices k_i and k'_i in the i-th dimension:

$$\mathcal{K}(k_i, k'_i) = \exp\Big(- \gamma * ||E[k_i] - E[k'_i]||_2^2 \Big) \tag{17}$$

Given a series X at index $(\mathbf{k}, t)$, for each dimension i, we compute the kernel-weighted sum of measure values as

$$U_{(\mathbf{k},i),t} = \frac{\sum_{\mathbf{k}' \in \text{Sib}(\mathbf{k},i)} X_{\mathbf{k}',t} \mathcal{K}(k_i, k'_i) A_{\mathbf{k}',t}}{\sum_{\mathbf{k}' \in \text{Sib}(\mathbf{k},i)} \mathcal{K}(k_i, k'_i) A_{\mathbf{k}',t}} \tag{18}$$

where $A_{\mathbf{k}',t} = 1$ for non-missing indices and 0 for missing indices.

When a dimension i is large, we make the above computation efficient by pre-selecting the top L members based on their kernel similarity.

We also compute two other measures: Sum of kernel weights and the variance in X values along each sibling dimension:

$$W_{(\mathbf{k},i),t} = \sum_{\mathbf{k}' \in \text{Sib}(\mathbf{k},i)} \mathcal{K}(k_i, k'_i) A_{\mathbf{k}',t} \tag{19}$$

$$V_{(\mathbf{k},i),t} = Var(X_{\text{Sib}(\mathbf{k},i),t}) \tag{20}$$

The last layer of the kernel-regression module is concatenation of U, V, and W components:

$$\mathbf{h}^{\text{kr}} = \text{Concat}(U_{(\mathbf{k},i),t}, V_{(\mathbf{k},i),t}, W_{(\mathbf{k},i),t}) \tag{21}$$

where $\mathbf{h}^{\text{kr}} \in \mathbb{R}^{3n}$.

procedure DeepMVI(X, A, M)
  TrainData=$(\mathbf{k}_i, t_i) \in A$, A^i=random misses around $(\mathbf{k}_i, t_i)$.
  model ← CreateModel() `/* Sec. 4.3 */`
  for iter = 0 to MaxIter **do**
    for each $(\mathbf{k}_i, t_i, A^i) \sim$ Batch(TrainData) **do**
      $\hat{X}_{\mathbf{k}_i,t_i} \leftarrow$ ForwardPass($X, A^i, \mathbf{k}_i, t_i$)
    Update model parameters Θ.
    Evaluate validation data for early stopping.
  `/* Impute test-blocks */`
  $\hat{X} \leftarrow$ ForwardPass($X, A, \bullet, \bullet$) over all test blocks in $\mathcal{I}(M)$.
return $\hat{X}$
procedure ForwardPass(X, A, $\mathbf{k}$, t)
  $\mathbf{h}^{\text{tt}}, \mathbf{h}^{\text{fg}} = TT(X, A)$.
  $\mathbf{h}^{\text{kr}} = KR(X, A)$. `Section 4.2`
return $\mathbf{h}^{\text{tt}}$, $\mathbf{h}^{\text{fg}}$, $\mathbf{h}^{\text{kr}}$
procedure TT(X, A)
  Index of the block containing time t is $j = T\%w$.
  $Y_j = W_f X_{jw:(j+1)w} + b_f$.
  `/*Similarly compute` Y_{j-1} `and` Y_{j+1}`.*/`
  Compute Query, Keys, and Values using Equations 8, 9, 10.
  Calculate Attn(Q(·), K(·), V(·), A, j) using Eqn. 11.
  Calculate multi-head attention:

$$\mathbf{h}_j = [\text{Attn}^1(\cdots), \ldots, \text{Attn}^{n_{\text{head}}}(\cdots)]$$

  Compute vector $\mathbf{h}^{\text{tt}}_j$ using Equations 13 and 14.
  $\mathbf{h}^{\text{tt}} = \mathbf{h}^{\text{tt}}_j[t\%w]$.
  Compute the fine grained attention vector:

$$\mathbf{h}^{\text{fg}} = FG_\theta(X, A) \texttt{/* Eqn 15 */}$$

return $\mathbf{h}^{\text{tt}}$, $\mathbf{h}^{\text{fg}}$
procedure KR(X, A)
  Compute the vectors $U_\bullet$, $W_\bullet$, and $V_\bullet$ (Equations 18, 19, 20).
  $\mathbf{h}^{\text{kr}} = \text{Concat}(U_{(\mathbf{k},i),t}, V_{(\mathbf{k},i),t}, W_{(\mathbf{k},i),t})$.
return $\mathbf{h}^{\text{kr}}$

Figure 3: The DeepMVI training and imputation algorithm

4.3 Network Parameters and Hyper-parameters

The parameters of the network span the temporal transformer, the embeddings of members of dimensions used in the kernel regression, and the parameters of the output layer. We use Θ to denote all the trainable parameters in all modules:

$$\Theta = \{W_f, b_f, W_q, b_q, W_k, b_k, W_v, b_v, W_{d_1}, W_{d_2}, W_d, b_d, \boldsymbol{w}_o, \boldsymbol{b}_o, E[m_{\bullet,\bullet}]\}.$$

These parameters are trained using the training objective described in Section 3 on the available data. Any off-the-shelf stochastic gradient method can be used for solving this objective. We used Adam with a learning rate 1e-3.

Network Hyper-parameters: Like any deep learning method, our network also has hyper-parameters that control the size of the network, which in turn impacts accuracy in non-monotonic ways.

Table 1: Datasets: All except the last two have one categorical dimension. Qualitative judgements on the repetitions of patterns along time and across series appear in the last two columns.

Dataset	Number of TS	Length of TS	Repetitions within TS	Relatedness across series
AirQ	10	1k	Moderate	High
Chlorine	50	1k	High	High
Gas	100	1k	High	Moderate
Climate	10	5k	High	Low
Electricity	20	5k	High	Low
Temperature	50	5k	High	High
Meteo	10	10k	Low	Moderate
BAFU	10	50k	Low	Moderate
JanataHack	76*28	134	Low	High
M5	10*106	1941	Low	Low

Many techniques exist for automatically searching for optimal values of hyper-parameters [1] based on performance on a validation set. These techniques are applicable in our model too but we refrained from using those for two reasons: (1) they tend to be computationally expensive, and (2) we obtained impressive results compared to almost all existing methods in more than 50 settings without extensive dataset specific hyper-parameter tuning. This could be attributed to our network design and robust training procedure. That said, in specific vertical applications, a more extensive tuning of hyper-parameters using any of the available methods [1] could be deployed for even larger gains.

The hyper-parameters and their default values in our network are: the number of filters $p = 32$ that controls the size of the first layer of the temporal transformer, the window size w of the first convolution layer. The hyper-parameter w also determines the size of the context key used for attention. If w is very small, the context size may be inadequate, and if it is too large compared to the size of each series we may over-smooth patterns. We use 10 by default. When the average size of a missing block is large (> 100) we use $w = 20$ to gather a larger context. The number of attention heads n_{head} is four and embedding size d_i is taken to be 10 in all our experiments.

5 EXPERIMENTS

We present results of our experiments on ten datasets under four different missing value scenarios. We compare imputation accuracy of several methods spanning both traditional and deep learning and approaches in Sections 5.3 and 5.4. We then perform an ablation study to evaluate the various design choices of DeepMVI in Section 5.5. In Section 5.6 we compare different methods on running time. Finally, in Section 5.7 we highlight the importance of accurate imputation algorithms on downstream analytics.

5.1 Experiment Setup

5.1.1 Datasets. We experiments on eight datasets used in earlier papers on missing value imputation [12]. In addition, due to the lack of multi-dimensional datasets in previous works, we introduce two new datasets, "JanataHack", "M5". Table 1 presents a summary along with qualitative judgements of their properties.

AirQ brings air quality measurements collected from 36 monitoring stations in China from 2014 to 2015. AirQ time series contain both repeating patterns and jumps, and also strong correlations across time series. Replicating setup [12], we filter the dataset to get 10 time series of 1000 length.

Chlorine simulates a drinking water distribution system on the concentration of chlorine in 166 junctions over 15 days in 5 minutes interval. This dataset contains clusters of similar time series which exhibit repeating trends.

Gas shows gas concentration between 2007 and 2011 from a gas delivery platform of ChemoSignals Laboratory at UC San Diego.

Climate is monthly climate data from 18 stations over 125 locations in North America between 1990 and 2002. These time series are irregular and contain sporadic spikes.

Electricity is on household energy consumption collected every minute between 2006 and 2010 in France.

Temperature contains temperature from climate stations in China from 1960 to 2012. These series are highly correlated.

MeteoSwiss is weather from different Swiss cities from 1980 to 2018 and contains repeating trends with sporadic anomalies.

BAFU consists of water discharge data by the BundesAmt Für Umwelt (BAFU), collected from Swiss rivers from 1974 to 2015. These time series exhibit synchronized irregular trends.

JanataHack is a multidimensional time series dataset[2] which consists of sales data spanning over 130 weeks, for 76 stores and 28 products (termed "SKU").

Walmart M5 made available by Walmart, involves the daily unit sales of 3049 products sold in 10 stores in the USA spanning 5 years. Since most of the 3,049 items have 0 sales, we retain the 106 most selling items averaged over stores. This gives us a 2 dimensional data of sales of 106 items across 10 stores.

5.1.2 Missing Scenarios Description. We experiment with four missing scenarios[12] considered to be the common missing patterns encountered in real datasets. Here we consider continuous chunks of missing values termed as blocks. We also consider a scenario with point missing values scattered throughout the dataset in Sec 5.5.

Missing Completely at Random (MCAR) Each incomplete time series has 10% of its data missing. The missing data is in randomly chosen blocks of constant size 10. We experiment with different % of incomplete time series.

Missing Disjoint (MissDisj) Here we consider disjoint blocks to be missing. Block size is T/N, where T is the length of time series, and N is the number of time series. For ith time series the missing block ranges from time step $\frac{iT}{N}$ to $\frac{(i+1)T}{N} - 1$, which ensures that missing blocks do not overlap across series.

Missing Overlap (MissOver) A slight modification on MissDisj, MissOver has block size of $2 * T/N$ for all time series except the last one for which the block size is still T/N. For the i-th time series the

[2]https://www.kaggle.com/vin1234/janatahack-demand-forecasting-analytics-vidhya

missing block ranges from time step $\frac{iT}{N}$ to $\frac{(i+2)T}{N} - 1$, which causes an overlap between missing blocks of series i with $i-1$ and $i+1$

Blackout considers a scenario where all time series have missing values for the same time range. Given a block size s all time series have values missing from t to $t+s$, where t is fixed to be 5%. We vary the block size s from 10 to 100.

5.1.3 **Methods Compared.** We compare with methods from both conventional and deep learning literature.
CDRec[11] is one of the top performing recent Matrix Factorisation based technique which uses iterative Centroid Decomposition.
DynaMMO[14] is a probabilistic method that uses Kalman Filters to model co-evolution of subsets of similar time series.
TRMF [28] is a matrix factorisation augmented with an auto-regressive temporal model.
SVDImp [24] is a basic matrix factorisation based technique which imputes using top k vectors in SVD factorisation.
BRITS [4] is a recent Deep learning techniques that uses a Bidirectional RNN that takes as input all the series' values at time t.
GPVAE [27] a deep learning method that uses Gaussian process in the low dimensional latent space representation of the data. GPVAE uses Variational Autoencoder to generate the imputed values in the original data space.
Transformer [25] is a deep learning method that uses a multi-head self-attention based architecture to impute the missing values in time-series.

5.1.4 Other Experiment Details.

Platforms Our experiments are done on the Imputation Benchmark[3] for comparisons with conventional methods. The benchmark lacks in support for deep learning based algorithms hence we compare our numbers for those outside this framework.

Evaluation metric We use Mean Absolute Error as our evaluation metric.

5.2 Visual Comparison of Imputation Quality

We start with a visual illustration of how DeepMVI's imputations compare with those of two of the best performing existing methods: CDRec and DynaMMO. In Figure 4, we visualize the imputations for different missing blocks on the Electricity dataset. First row is for MCAR scenario whereas second row is for Blackout scenario. First observe how DeepMVI(Blue) correctly captures both the shape and scale of actual values (Black) over a range of missing blocks. On the MCAR scenario CDRec gets the shape right, only in the first and fourth blocks, however it is off with scale. In the Blackout scenario, CDRec only linearly interpolates the values in missing block, whereas DynaMMO is only slightly inclined towards ground-truth. However, both CDRec and DynaMMO miss the trend during Blackout whereas DeepMVI successfully captures it because of careful pattern match within a series.

5.3 Comparison on Imputation Accuracy

Given the large number of datasets, methods, missing scenarios and missing sizes we present our numbers in stages. First in Figure 5

[3]https://github.com/eXascaleInfolab/bench-vldb20

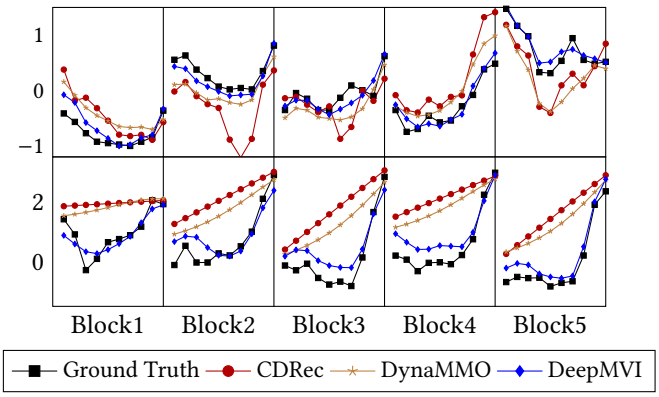


Figure 4: Visualised Imputations on Electricity Dataset. The top row shows MCAR missing blocks while the bottom rows is for Blackout scenario.

we show comparisons in MAE of all conventional methods on five datasets under a fixed x =10% of series with missing values in MCAR, MissDisj, MissOver and all series in Blackout with a block size of 10. Then, in Figure 6 we show more detailed MAE numbers on three datasets (AirQ, Climate and Electricity) where we vary the percent of series with missing values (x) from 10 to 100 for MCAR, MissDisj, MissOver and the block size from 10 to 100 in Blackout. From these comparisons across eight datasets we make the following observations:

First, observe that DeepMVI is better or comparable to all other methods under all missing values scenarios and all datasets. Our gains are particularly high in the Blackout scenario seen in the last column in graphs in Figure 6 and in the bottom-right graph of Figure 11. For accurate imputation in Blackouts, we need to exploit signals from other locations of the same series. Matrix factorisation based methods such as SVDImp and TRMF fail in doing so and rely heavily on correlation across time series. TRMF's temporal regularisation does not seem to be helping in capturing long term temporal correlations. DynaMMO and CDRec are able to capture within time series dependencies better than matrix factorisation methods. But they are still much worse than DeepMVI, particularly on Gas in Figure 11, and Climate, Electricity in Figure 6.

In the MissDisj/MissOver scenario where the same time range is not missing across all time series, methods that effectively exploit relatedness across series perform better on datasets with highly correlated series such as Chlorine and Temp. Even in these scenarios we provide as much as 50% error reduction compared to existing methods.

MCAR is the most interesting scenario for our analysis. Most of the baselines are geared towards capturing either inter or intra TS correlation but none of them are able to effectively combine and exploit both. MCAR owing to small block size and random missing position can benefit from both inter and intra correlation which are fully exploited by our model. DeepMVI achieves strictly better numbers than all the baselines on all the datasets. For Climate and Electricity datasets, we reduce errors between 20% and 70% as seen in the first column of Figure 6.

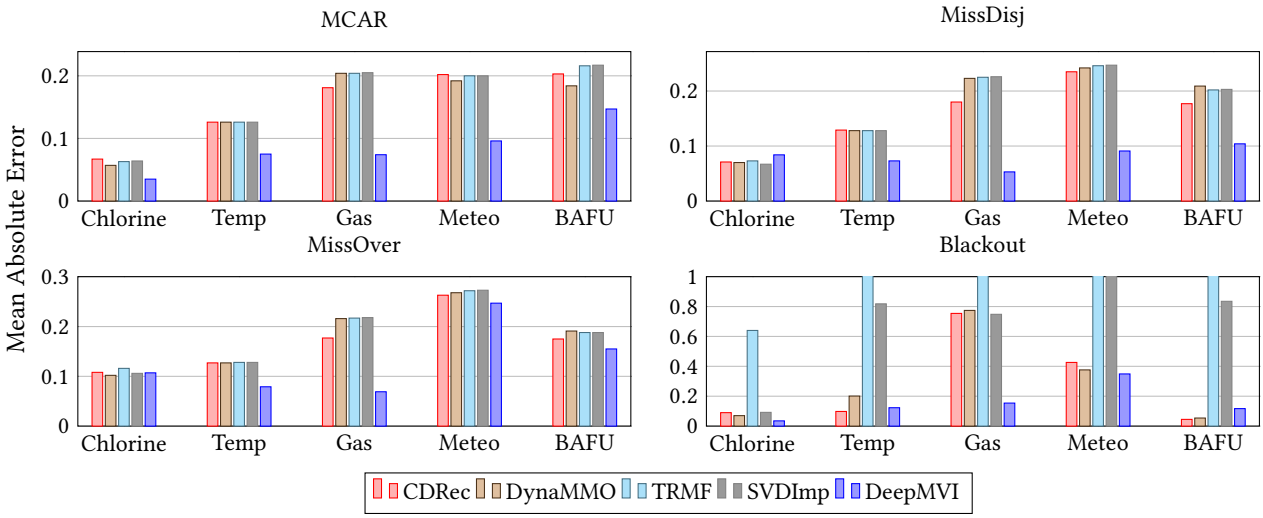


Figure 5: Mean Absolute Errors (y-axis) on five other datasets (on x-axis) on all four scenarios – MCAR, MissDisj, MissOver, and Blackout. Here, a fixed $x = 10\%$ of the series in each dataset has missing blocks.

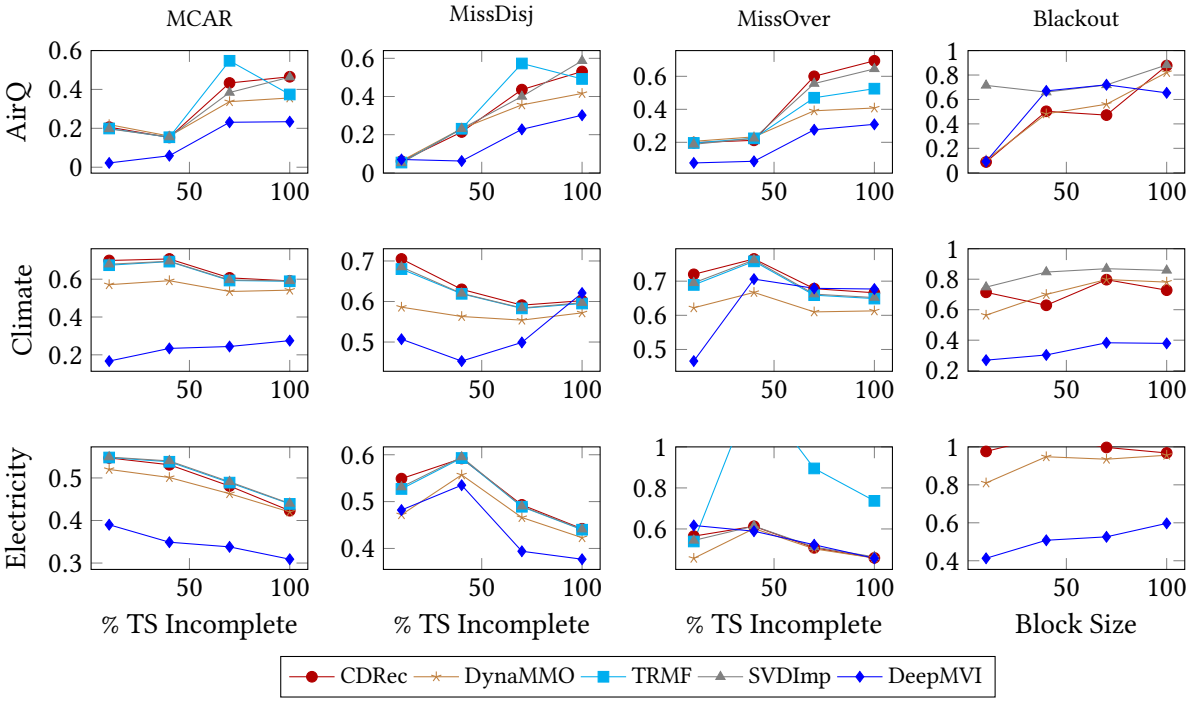


Figure 6: Mean Absolute Errors (y-axis) on three datasets along rows and under four missing scenarios along columns. X-axis is percent of time-series with a missing block for MCAR, MissDisj, MissOver and size of the missing block for Blackout.

5.4 Comparison with Deep Learning Methods

We compare our method with with two state-of-the-art deep learning imputation models, along with a vanilla transformer model. We use the official implementation of BRITS and GP-VAE to report these numbers. We present MAE numbers in Table 2. First consider the comparison on the two multi-dimensional datasets: M5 and JanataHack. Both have store and items as the two dimensions in addition to time (Table 1). We experiment in the MCAR scenario with $x = 100\%$ time-series with a missing block. We find that DeepMVI outperforms all the other imputation models on both these datasets.

Table 2: Comparison with Deep Learning Methods. Blackout has missing blocks of size 100, and MCAR has missing values in 100% of the time series.

Model	Walmart M5	JantaHack	Climate		Electricity		Meteo	
	MCAR	MCAR	MCAR	Blackout	MCAR	Blackout	MCAR	Blackout
BRITS [4]	0.69	0.22	0.26	0.69	0.28	1.16	0.19	0.77
GPVAE[8]	0.60	0.28	0.43	0.81	0.33	1.08	0.26	1.31
Transformer [25]	0.56	0.24	0.29	0.67	0.36	0.97	0.29	0.48
DeepMVI(Ours)	0.53	0.16	0.28	0.38	0.31	0.60	0.20	0.46

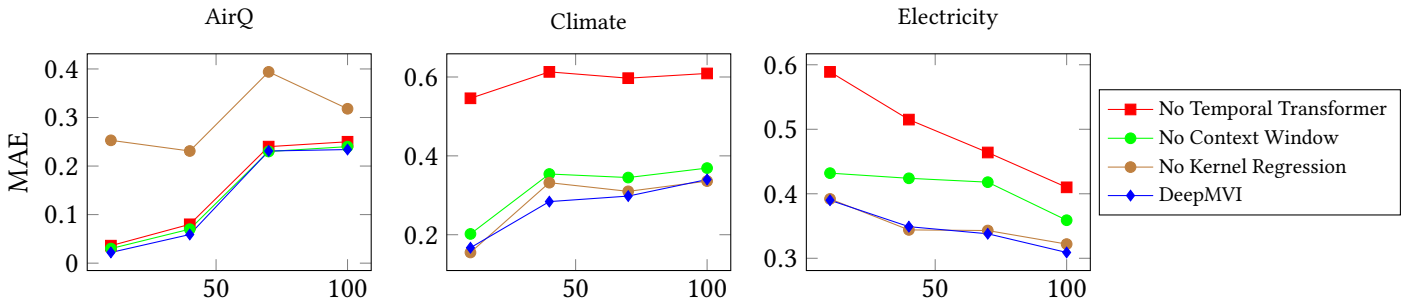


Figure 7: Ablation Study done via 3 datasets represented by different plots AirQ,Climate,Electricity on MCAR scenario. y-axis shows MAE and x-axis % of Missing TS.

The decrease in MAE is especially significant for JantaHack which has high correlations across different stores for given products.

We next present our numbers on Climate, Electricity and Meteo, on MCAR and Blackout. Here too DeepMVI is either the best or close to the best in all dataset-scenario combinations. On the Blackout scenario our method is significantly better than BRITS, the state-of-the-art. We attribute this to our method of creating artificial blackouts around training indices. In contrast, the BRITS model depends too much on immediate temporal neighborhood during training. We see that the Transformer model can capture periodic correlations within time series such as those in Climate MCAR. However it fails to capture more subtle non-periodic repeating pattern which requires attention on window feature vectors. Such patterns are prevalent in Electricity and Meteo datasets.

5.5 Justifying Design Choices of DeepMVI

DeepMVI introduces a Temporal transformer with an innovative left-right window feature to capture coarse-grained context, a fine-grained local signal, and a kernel regression module that handles multi-dimensional data. Here we perform an ablation study to dissect the role of each of these parts.

5.5.1 Role of Context Window Features. We study the role of query and key used in our Temporal Transformer module. Our query/key consists of concatenated window features of previous and next block arithmetically added with positional encoding. Positional encoding encode the relative positions and have no information pertaining to the context of the block where imputation needs to be performed. A question to be asked here was whether the contextual information around a missing block help in a better attention mechanism or whether the attention mechanism just ignores this contextual information doing a fixed periodic imputation. Figure 7 shows this method (Green). These experiments are on MCAR and x-axis is increasing % of missing TS. Comparing the green and blue, we see that our window context features did help on two of the three datasets, with the impact on Electricity being quite significant. This might be attributed to the periodic nature of the climate dataset compared to non-periodic but strongly contextual information in electricity.

5.5.2 Role of Temporal Transformer and Kernel Regression. In Figure 7 we present error without the Temporal Transformer Module(Red) and without Kernel Regression Module(Brown). We see some interesting trends here. On Climate and Electricity where each series is large (5k) with repeated patterns across series, we see that dropping the temporal transformer causes large jumps in error. On Climate error jumps from 0.15 to 0.55 with 10% missing! In AirQ we see little impact. However, on this data dropping Kernel regression causes a large increase in error jumping from 0.04 to 0.25 on 10% missing. Kernel regression does not help much beyond temporal transformer on Climate and Electricity. These experiments show that DeepMVI is capable of combining both factors and determining the dominating correlation via the training process.

5.5.3 Role of Fine-Grained Local Signal. (Equation 15). This signal is most useful for small missing blocks. Hence we modify the MCAR missing scenario such that missing percentage from all time series is still 10%, but the missing block size is varied from 1 to 10. Figure 8 shows the results where we compare our MAE with and without fine grained local signal with CDRec algorithm on the Climate Dataset. The plot shows that including fine grained signal helps improve accuracy over a model which ignores the local information.

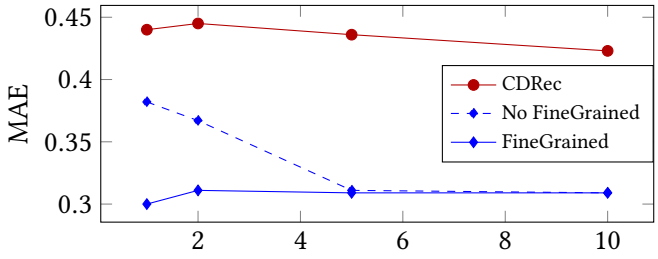


Figure 8: MAE (y-axis) vs missing block size (x-axis) with 10% missing on Climate Dataset. Gains from fine-grained local signal decreases with increasing block size.

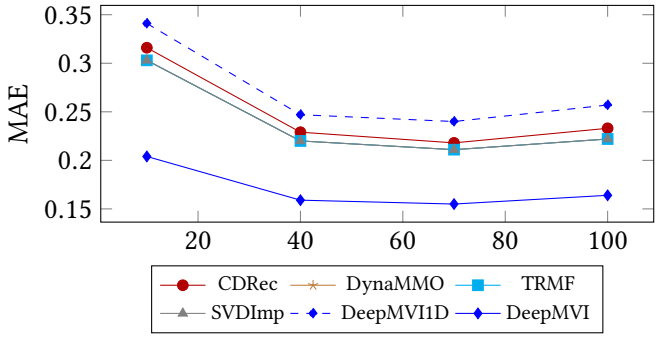
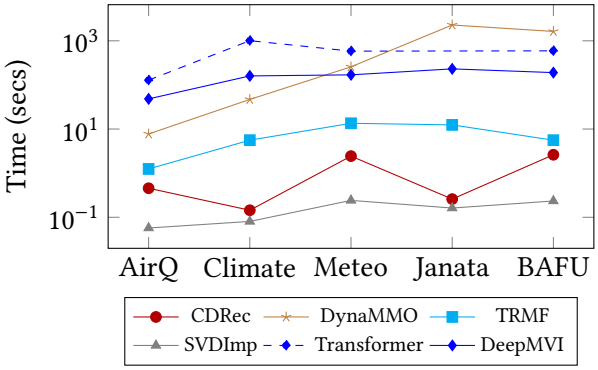
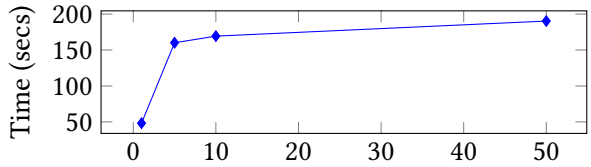


Figure 9: MAE (y-axis) for MCAR scenario on JanataHack. X-axis is percent of time series with missing blocks.



(a) Absolute Runtime (*s*) on y-axis with dataset arranged in increasing data set size. MCAR scenario with all time series having missing blocks (x=100%).



(b) Absolute Runtime (*s*) for DeepMVI on y-axis with X-axis showing series length in factor of 1K. The numbers are on real datasets (AirQ,Climate,Meteo,BAFU). Number of time series is 10. More details in Section 5.6

Figure 10: Runtime Analysis of DeepMVI

Also the gain in accuracy with fine grained local signal diminishes with increasing block size which is to be expected.

5.5.4 Effect of multidimensional kernel regression. For this task, we run two variants of our model. The first model dubbed as DeepMVI1D flattens the multidimensional index of time series by getting rid of the store and product information. The second variant is the proposed model itself which retains the multi-dimensional structure and applies kernel embeddings in two separate spaces. In DeepMVI each time series is associated with two embeddings of size k each. To keep the comparison fair, DeepMVI1D uses embedding of size $2k$. Since other methods have no explicit model for multi-dimensional indices, the input is a flattened matrix, similar to DeepMVI1D.

Figure 9 shows the performance of the variants compared to the baselines on MCAR for increasing percentage x of number of series with a missing block. Observe how in this case too DeepMVI is significantly more accurate than other methods including DeepMVI1D. If each series is small and the number of series is large, there is a greater chance of capturing spurious correlation across series. In such cases, the external multidimensional structure that DeepMVIexploits helps to restrict relatedness only among siblings of each dimension. We expect this difference to get magnified as the number of dimensions increase.

5.6 Running Time

The above experiments have shown that DeepMVI is far superior to existing methods on accuracy of imputation. One concern with any deep learning based solution is its runtime overheads. We show in this section that while DeepMVI is slower than existing matrix factorization methods, it is more scalable with TS length and much faster than off-the-shelf deep learning methods.

We present running times on AirQ, Climate, Meteo, BAFU, and JanataHack datasets in Figure 10a. The x-axis shows the datasets ordered by increasing total size and y-axis is running time in log-scale. In addition to methods above we also show running time with an off the shelf transformer method. Matrix factorisation based method like CDRec and SVDImp are much faster than DynaMMO and DeepMVI. But compared to the vanilla Transformer our running time is a factor of 2.5 to 7 smaller. The running time of DynaMMO exceeds the running time of other algorithms by a factor of 1000 and increases substantially with increasing series length, which undermines the accuracy gains it achieves. On the JanataHack dataset, DynaMMO took 25 mins (1.5e9 μs) compared to DeepMVI which took just 2.5 mins.

We next present our numbers on scalability of DeepMVI in Figure 10b. The x axis denotes the length of times series in factors of 1K. The points correspond to datasets AirQ, Climate, Meteo and BAFU for 1K, 5K, 10K and 50K respectively. All these datasets have 10 time series. We can see a sub-linear growth of running time with the series length. An intuition behind the same is that our training algorithm learns patterns, which in case of seasonal time series can be learnt by seeing a small number fraction of the series, abstractly one season worth of data.

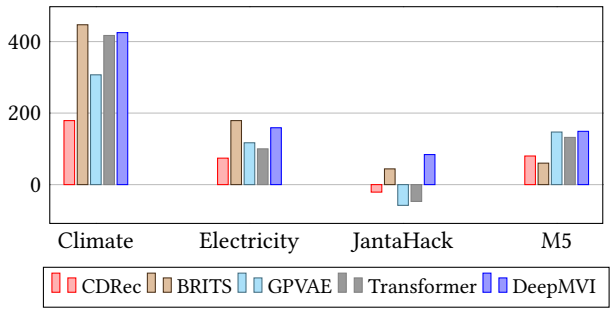


Figure 11: Difference between Mean Absolute Error of DropCell and each algorithm is on y-axis, and four other datasets on x-axis.

5.7 Impact on downstream analytics

A major motivation for missing value imputation is more accurate data analytics on time series datasets [3, 10, 18, 21]. Analytical processing typically involves studying trends of aggregate quantities. When some detailed data is missing, a default option is to just ignore the missing data from the aggregate statistic. Any MVI method to be of practical significance in data analysis should result in more accurate top-level aggregates than just ignoring missing values. We present a comparison of the different MVI methods on the average over the first dimension so the result is a $n-1$ dimensional aggregated time series. Except in JanataHack and M5, this results in a single averaged time series.

Apart from computing the above statistic on the imputation output by various algorithms, we also compute this statistic with just the missing values in the set dropped from the average. We call this the DropCell method. We consider four datasets: Climate, Electricity, JanntaHack, and M5, each in MCAR with 100% of the time series containing missing values. On each of these datasets, we first compute the aggregate statistic using true values. For Climate and Electricity, this returns a single series with value at time t as the average of values at all series at time t. For JantaHack, we average over 76 stores resulting in average sales of 28 products. Similarly on M5, we average over 10 stores giving us sales of 106 items. Next we compute the aggregate statistic with missing values imputed by five algorithms: CDRec, BRITS, GPVAE, Transformer, and DeepMVI. We compute MAE between aggregate with imputed values and aggregate over true values.

In Figure 11, we report the difference between MAE of DropCell and MAE of the algorithm. We see that on the JanataHack dataset, three existing imputation method CDRec, GPVAE, and Transformer provide worse results than just dropping the missing value in computing the aggregate statistic. In contrast DeepMVI provides gains over this default in all cases. Also, it is overall better than existing methods, particularly on the multidimensional datasets. This illustrates the impact of DeepMVI on downstream data analytics.

6 CONCLUSION AND FUTURE WORK

In this paper, we propose DeepMVI, a deep learning method for missing value imputation in multi-dimensional time-series data. DeepMVI combines within-series signals using a novel temporal transformer, across-series signals using a multidimensional kernel regression, and local fine-grained signals. The network parameters are carefully selected to be trainable across wide ranges of data sizes, data characteristics, and missing block pattern in the data.

We extensively evaluate DeepMVI on ten datasets, against comparing seven conventional and three deep learning methods, and with five missing-value scenarios. DeepMVI achieves up to 70% error reduction compared to state of the art methods. Our method is up to 50% more accurate and six times faster than using off-the-shelf neural sequence models. We also justify our module choices by comparing DeepMVI with its variants. We show that DeepMVI's performance on downstream analytics tasks is better than dropping the cells with missing values as well as existing methods.

Future work in this area includes applying our neural architecture to other time-series tasks including forecasting.

REFERENCES

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[2] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. 2010. A singular value thresholding algorithm for matrix completion. *SIAM Journal on optimization* 20, 4 (2010), 1956–1982.

[3] José Cambronero, John K. Feser, Micah J. Smith, and Samuel Madden. 2017. Query Optimization for Dynamic Imputation. *Proc. VLDB Endow.* 10, 11 (2017).

[4] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. 2018. Brits: Bidirectional recurrent imputation for time series. *arXiv preprint arXiv:1805.10572* (2018).

[5] Prathamesh Deshpande and Sunita Sarawagi. 2019. Streaming adaptation of deep forecasting models using adaptive recurrent units. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1560–1568.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).

[7] Valentin Flunkert, David Salinas, and Jan Gasthaus. 2017. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *CoRR* abs/1704.04110 (2017).

[8] Vincent Fortuin, Dmitry Baranchuk, Gunnar Rätsch, and Stephan Mandt. 2020. Gp-vae: Deep probabilistic time series imputation. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1651–1661.

[9] Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks* 18, 5-6 (2005), 602–610.

[10] Sean Kandel, Ravi Parikh, Andreas Paepcke, Joseph M Hellerstein, and Jeffrey Heer. 2012. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*. 547–554.

[11] Mourad Khayati, Philippe Cudré-Mauroux, and Michael H Böhlen. 2019. Scalable recovery of missing blocks in time series with high and low cross-correlations. *Knowledge and Information Systems* (2019), 1–24.

[12] Mourad Khayati, Alberto Lerner, Zakhar Tymchenko, and Philippe Cudré-Mauroux. 2020. Mind the gap: an experimental evaluation of imputation of missing values techniques in time series. *Proceedings of the VLDB Endowment* 13, 5 (2020), 768–782.

[13] Jason Li, Vitaly Lavrukhin, Boris Ginsburg, Ryan Leary, Oleksii Kuchaiev, Jonathan M Cohen, Huyen Nguyen, and Ravi Teja Gadde. 2019. Jasper: An end-to-end convolutional neural acoustic model. *arXiv preprint arXiv:1904.03288* (2019).

[14] Lei Li, James McCann, Nancy S Pollard, and Christos Faloutsos. 2009. Dynammo: Mining and summarization of coevolving sequences with missing values. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 507–516.

[15] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. 2019. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Advances in Neural Information Processing Systems*. 5243–5253.

[16] Roderick JA Little and Donald B Rubin. 2002. Single imputation methods. *Statistical analysis with missing data* (2002), 59–74.

[17] Yukai Liu, Rose Yu, Stephan Zheng, Eric Zhan, and Yisong Yue. 2019. NAOMI: Non-autoregressive multiresolution sequence imputation. In *Advances in Neural Information Processing Systems*. 11238–11248.

[18] Chris Mayfield, Jennifer Neville, and Sunil Prabhakar. 2010. ERACER: A Database Approach for Statistical Inference and Data Cleaning. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*.

[19] Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. 2010. Spectral regularization algorithms for learning large incomplete matrices. *The Journal of Machine Learning Research* 11 (2010), 2287–2322.

[20] Jiali Mei, Yohann De Castro, Yannig Goude, and Georges Hébrail. 2017. Nonnegative matrix factorization for time series recovery from a few temporal aggregates. In *International Conference on Machine Learning*. PMLR, 2382–2390.

[21] Tova Milo and Amit Somech. 2020. Automating exploratory data analysis via machine learning: An overview. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2617–2622.

[22] David Salinas, Michael Bohlke-Schneider, Laurent Callot, Roberto Medico, and Jan Gasthaus. 2019. High-dimensional multivariate forecasting with low-rank Gaussian Copula Processes. In *Advances in Neural Information Processing Systems*. 6827–6837.

[23] Rajat Sen, Hsiang-Fu Yu, and Inderjit Dhillon. 2019. Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. *arXiv preprint arXiv:1905.03806* (2019).

[24] Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B Altman. 2001. Missing value estimation methods for DNA microarrays. *Bioinformatics* 17, 6 (2001), 520–525.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NIPS*.

[26] Kevin Wellenzohn, Michael H Böhlen, Anton Dignös, Johann Gamper, and Hannes Mitterer. 2017. Continuous imputation of missing values in streams of pattern-determining time series. (2017).

[27] Jinsung Yoon, William R Zame, and Mihaela van der Schaar. 2018. Estimating missing data in temporal data streams using multi-directional recurrent neural networks. *IEEE Transactions on Biomedical Engineering* 66, 5 (2018), 1477–1490.

[28] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S Dhillon. 2016. Temporal regularized matrix factorization for high-dimensional time series prediction. In *Advances in neural information processing systems*. 847–855.