Missing Data Imputation for Industrial Big Data

Pedro Allen Revez
pedroallen@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

September 2020

Abstract

Missing Data is a problem that is felt by all Machine Learning practitioners. Ranging from simple statistical solutions, to complex Artificial Intelligence approaches, research and academia has mainly shown that there is no correct solution when solving this problem. It is context dependent - on the type of data, type of problem, and even domain knowledge. Missing Data is still an open problem.

This dissertation proposes a scalable, lightweight Autoencoder model for solving Missing Data in Industrial Big Data. By combining modern Deep Learning approaches, and historically proven methods to handle Time-Series data - Fourier Transformations - the proposed model is a synergistic approach for Industrial Problems. We provide a generic framework of evaluation for Imputation methods, tailored for Time-Series data.

The results on Wind-Turbine and Household Electrical Consumption data, show the ability of the proposed model to handle Missing Data in different data conditions, while providing structured and expressive representations for Time-Series data.

Keywords: Deep Learning, Industrial Big Data, Fourier Transformations, Missing Data

1. Introduction

An A.I. based company is heavily dependent on the data it is using for producing value for any customer. Therefore, Missing Imputation is a problem that should not be overlooked, in order to ensure the best quality data is provided for building better Intelligence. Often, modern Deep Learning (DL) and Machine Learning (ML) methods are not impervious to incomplete data, and requires a set of complete data in order to be able to function as intended.

Industrial Sensors are known to be prone to failures. The origin and root of Missing Data is context dependent, on the type of problem, type of data, or other external factors. As the state-of-the-art moves towards ML and DL methods, we focus our research to solve the Missing Data problem, using DL methods. Solving Missing Data is an **open problem**, and the main motivation of this dissertation is to find a proper solution applied to **Industrial Processes**.

We propose an imputation method for Multi-Variate Time-Series that is scalable and efficient for large Industrial applications. Below are outlined the main contributions of this thesis.

1. Generic Imputation Evaluation Framework for Time-Series, that takes into account both Missing Rate and data missing in a sequence
2. Missing Data Imputation for Multi-Variate Time-Series
3. A new class of Autoencoder model based on Short Time Fourier Transformations
4. Application in Industrial Big Data

2. Background

2.1. Missing Data

During the collection procedure of data, errors might occur. Some observations in the data will be missing - **Missing Data**. Moreover, Missing Data can originate under three distinct mechanisms.

2.1.1 Not a Number - NaN

In 1985 the IEEE754 standard introduced the use of the Not A Number (NaN) data type [1]. It is a data type that can be interpreted as undefined or unrepresentable. Initially introduced to handle floating-point arithmetic, it was also adopted to represent Missing Data in other data types, i.e. text.

Operating any value with a NaN will result in a NaN value. The NaN value propagates it's undefinition onto other values. The implications of this behaviour in the arithmetic means that any statistical model or Machine Learning method that operated numeric values and encouters a NaN value, all the following computations that depend on this value will be NaN as well.

2.1.2 Missing Completely at Random - MCAR

When missing data occurs due to circumstances that are **external** or **unrelated** to any of the past observed or unobserved data, it is said that the data is Missing Completely at Random [12].

2.1.3 Missing at Random - MAR

When Missing Data occurs because of **past observed behavior**, it is a Missing at Random [12]. These occur when the cause is unrelated to the value *itself*, but may be related to observed values of other variables.

2.1.4 Missing Not at Random - MNAR

When data is missing due to it's value, it is Missing Not at Random. [12], meaning that a value is missing given that the past data showed a **behaviour** that led to a missing value, or a behaviour that made it impossible to register any further values.

2.2. Autoencoders

In 1986 Rumelhart et al. [6] proposed the **Autoencoder** (AE) architecture. It is composed of an **Encoder** model and a **Decoder** model. The objective is to learn a *decoding function* g_Θ that is able to map latent-space representations $z \in Z$ to the *reconstructed* input $x' \in X'$. The representation z is obtained through an *encoding function* f_Φ.

Usually, latent-space z has lower-dimensionality than the input-space x. The network will be forced learn an **efficient compressed representation**, that can capture most structure. This is highly similar to PCA, as described by Hinton et. al [9].

2.3. Denoising Autoencoders

Since AE are trying to model an identity function, there is great risk of **overfitting**. Denoising Autoencoders (DAE) by Bengio et. al [18] are exactly what the name describes, it cleans noisy inputs.

By introducing a *stochastic corrupting function* C that will *partially corrupt* the input, it acts as a **regularization** agent that improves the *robustness* of the model, and avoids overfitting by deviating from the identity function. The corruption function $\tilde{x}_i = C_D(\tilde{x}_i|x_i)$ is applied over all the inputs, the *corruption* is drawn from distribution D, usually a Gaussian Distribution, with *intensity* σ.

The training procedure barely differs from a classic AE. The only difference is that the model optimizes the reconstruction of the original input, when it is given a corrupt input. Hence, the loss function of a DAE is:

$$L_{DAE}(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n}(x_i - g_\theta(f_\phi(\tilde{x}_i)))^2 \quad (1)$$

2.4. Fourier Transformations

The Fourier Transforms (FT) is a transformation on the **time-domain**, into a spectral or **frequency** domain. The objective of a FT, is to **decompose** a signal into a **linear combination** of **periodic** waveforms (frequency constituents), generally sinusoidal waves[15].

2.4.1 Discrete Fourier Transform

Unfortunately in the realm of computers, there is **no continuous** data. We are working with **discrete time** steps n. The Discrete Fourier Transform (DFT) is then defined by the discrete implementation of a FT [15]:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{N}nk}, k = 0, 1, \dots N-1 \quad (2)$$

The result of a DFT is a complex-valued number [19]. We can obtain a real-valued DFT, by using Euler's identity. For a N point signal, real and imaginary terms are **projected** into sine and cosine components. We get two resulting arrays of size $\frac{N}{2} + 1$, which can be synthesized to [15]:

$$X[k] = \sum_{n=0}^{N/2} X_{re}[n]\cos(\frac{2\pi kn}{N}) - X_{im}[n]\sin(\frac{2\pi kn}{N}) \quad (3)$$

2.4.2 Inverse Discrete Fourier Transform

The original signal can be recovered from the frequency components, using the **Inverse Discrete Fourier Transform** (IDFT):

$$x[n] = \frac{1}{N}\sum_{n=0}^{N-1} X[k]e^{j\frac{2\pi}{N}nk}, n = 0, 1, \dots N-1 \quad (4)$$

2.4.3 Windowing Function

In spectral analysis, it is customary to analyze just a short segment of the signal. Windowing functions are zero-valued except for the chosen interval of length N, usually symmetrical in the middle, where the maximum of the function is. Then the window *tapers* away from the center to the edges of the interval. A windowing function is then to be multiplied by the signal, as to obtain a segment of that signal, non-zero in the chosen interval. The **Hamming window** is one of such functions [15] and is defined by:

$$w(t) = 0.54 - 0.46cos(\frac{2\pi t}{T}), 0 \leq t \leq T \quad (5)$$

2.4.4 Short Time Fourier Transform

A Short Time Fourier Transform (STFT) is a DFT applied to a finite partition of the whole signal. The objective of a STFT is to observe the changes of the frequency domain over time. In order to obtain smooth transitions between the frequency domain of contiguous segments, the data is partitioned into **overlapping segments**, that are extracted every R steps (hop size)[**?**]. The STFT, approximates the true Fourier Transform [**?**], and as defined by Alen et. al [**?**] is:

$$X_m[\omega] = \sum_{n=-\infty}^{\infty} x[n]w(n - mR)e^{-j\omega n} \quad (6)$$

where w is a **windowing** function of length M, on a signal partition centered in time mR.

2.4.5 Overlap-Add

The Hamming Window is one of the most popular windowing functions, due to having the **Constant OverLap-Add** ($COLA$) property. A window $w(n)$ is said to be $COLA$ at hop-size $R, w \in COLA(R)$ i.f.f:

$$\sum_{m=-\infty}^{\infty} w(n - mR) = 1, \forall n \in \mathbb{Z} \quad (7)$$

If a windowing function has $COLA$ property at hop-size R, then the sum of **successive** DFTs over time, will equal the DFT of the whole signal $X[\omega]$ [15]. From equation 7, the sum of the DFTs of extracted partitions of length M is:

$$\sum_{m=-\infty}^{\infty} X_m[\omega] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x[n]w(n-mR)e^{-j\omega n} = \quad (8)$$

$$= \sum_{m=-\infty}^{\infty} x[n]e^{-j\omega n} \sum_{n=-\infty}^{\infty} w(n - mR) = \quad (9)$$

If the windowing function $w(n) \in COLA(R)$ then, the right-hand sum will be 1, by definition:

$$= \sum_{m=-\infty}^{\infty} x[n]e^{-j\omega n} = DFT_\omega(x) = X(\omega) \quad (10)$$

The Hamming Window is $COLA(M/2)$ which means that windows are to be extracted with a 50% overlap, in order to obtain the original signal from the sum of contiguous segments [15].

2.5. F-Principle

Deep Neural Networks (DNN) are often called "black-box" models - the internal workings are not fully known. Yanyang et al. explored DNN generalization capabilities in terms of the **frequency-space** - the **F-Principle**. The authors *empirically* found that for a general class of functions dominated by **low-frequencies**, a DNN would first generalize low-frequency components, and then slowly learn high-frequency components [13].

According to the principle, for a class of data dominated by low frequency components, the model's optimization can be stopped, as soon as it reaches a *plateau*. This avoids overfitting high-frequency components [21]. Moreover, if the data is not low-frequency component dominated, Early Stopping [4] is used for better generalization, not fitting noisy components.

3. Proposed Model

3.1. FFTron Autoencoder

The model receives as an input a window - a sequence of m points of a Multi-Variate Time-Series signal $x_{i,m} = (x_{i,1}, x_{i,2}, \cdot, x_{i,M})$. When training, a Gaussian corruption function is passed over the inputs. A **Hamming** windowing function is applied on the input window. The result of a DFT will only approximate the *true* FT. This leads to a phenomena called **spectral leakage** where harmonics of itself are *leaked*, because the data is a finite segment.

By *tapering* the window with a Hamming windowing function, we can attenuate the effect of spectral leakage. The sequence length of the window is a customizable **hyper-parameter** of the model, where one is trading frequency-domain resolution for time-domain resolution.

The Encoder applies DFT onto the windowed input data, and is then linearly mapped onto latent-space dimensions.

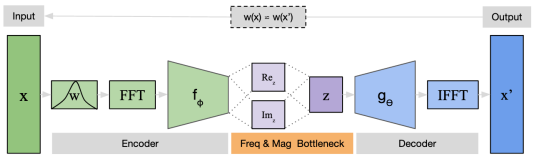

Figure 1: A diagram of the overall architecture of a FFTron Autoencoder.

The latent-space representation z should be vector composed of real and imaginary components $z = [Re(\vec{X}), Im(\vec{X})]_N$, due to the FT being a complex-valued transformation. However, Yao et.al [20] experimented with complex-valued Recurrent Neural Networks in order to handle FT computations, and observed that by **concatenating** two real-valued vectors of real and imaginary components the task could still be solved.

The Decoder is the **inverse operation** of the Encoder. It is composed of a linear mapping back into the original dimensions, and then a IDFT is applied to obtain the decoded signal in the time-domain.

The FFTron Autoencoder outputs a prediction that is **windowed** (tapered at the edges). Given

that **Hamming** window is $COLA(\frac{M}{2})$, the sum of overlapping segments (by 50%), **recovers** the original signal (Eq. 10). With the exception of the beginning and the end of the prediction, which will still be windowed due to not having an overlapping segment. These are manually inverted. Therefore, instead of outputting a batch of windows, the model produces one singular sequence.

A variation of the MSE optimization objective is used, by optimizing w.r.t. to the windowed version of the target. The output of the model is a windowed signal, which is used for overlapping predictions:

$$\mathcal{L}(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n}(w_i \cdot x_i - g_\theta(f_\phi(\tilde{x}_i)))^2 \qquad (11)$$

3.2. Imputation

Before using the model to impute, the data is required to be zero *initialized*. The proposed model will encode the input, and then decode the encoded input, to produce a reconstruction of the original signal. The original Missing Data will be **replaced** with the reconstructed signal.

Algorithm 1: Generalized Autoencoder Imputation Algorithm

Result: Ω
$\Gamma \leftarrow Read(\aleph)$
$\mathcal{M}_{id} \leftarrow GetMask(\Gamma)$
$\hat{\Gamma} \leftarrow Initialize(\Gamma)$
$\hat{\Gamma} \leftarrow Decode(Encode(\hat{\Gamma}))$
$\Omega \leftarrow Replace(\Gamma, \hat{\Gamma}, \mathcal{M}_{id})$

4. Experimental Setup

4.1. Data

The **Wind-Turbines** dataset comprises 8 sensor readings. Examples of a wind-turbine are *wind speed* measurements, *rotations per minute* on a mechanical piece, etc. All the measurements used in the dataset particular focus on a specific component inside the Wind-Turbine.

The second dataset is about individual house **electrical consumption**, provided by UCI Machine Learning, a popular repository of open-data [5]. The data contains minutely measurements of electrical power consumption, in an individual household.

All the datasets underwent the same preprocessing procedure. All the data was **standardized**, to diminish the effect of different scales that might bias the proposed algorithm. The datasets were split 80/20% into a **train** and **test** set.

4.2. Testing Framework

A Missing Regime is defined by two values: Missing Rate (how much data is missing) and a Missing Sequence (how many data points in a sequence are missing). The test-matrix comprises 100 different Missing Regimes. These regimes are the result of the combination of a range of Missing Rate from 10% to 60%, and Missing Sequences from 1 point to 100 points. Each experiment will be ran on a set of 20000 data points drawn from the test-set. Experiments across models are compared with one another, averaged across Missing Rate or Missing Sequence.

4.3. Evaluation Metrics

4.3.1 Mean Absolute Error

$$MAE(\hat{x}, x) = \frac{1}{N}\sum_{i=0}^{N}|\hat{x_i} - x_i| \qquad (12)$$

The **Mean Absolute Error** metric evaluates the residual error in absolute quantities, and is the magnitude of the error of a prediction, on average. It is a metric that is **robust to outliers**, given their expression on average [2].

4.4. Mean Squared Error

$$MSE(\hat{x}, x) = \frac{1}{N}\sum_{i=0}^{N}(\hat{x_i} - x_i)^2 \qquad (13)$$

The **Mean Squared Error** metric measures the squared error. In contrast to MAE, outlier residuals will have more expression [2].

4.5. Root Mean Squared Error

$$RMSE(\hat{x}, x) = \sqrt{\frac{1}{N}\sum_{i=0}^{N}|\hat{x_i} - x_i|} \qquad (14)$$

The **Root Mean Square Error** metric has the same shape as the MSE, but it is square rooted. This has the advantage of the metric being in the same units as the data, and therefore more **interpretable** metric [2].

4.6. R-Squared

$$R^2(\hat{x}, x) = 1 - \frac{\sum_{i=0}^{N}(\hat{x_i} - x_i)^2}{\sum_{i=0}^{N}(\bar{x_i} - x_i)^2} \qquad (15)$$

The **coefficient of determination** - R-squared (R2) metric - is useful for discerning whether a good performance is **biased** by being close to mean values. High values of R2 means the the model variance is similar to the variance in the data [2]. If the metric value is low the prediction variance correlates less with the data variance [2].

4.7. Inducing Missing Data

The only regime to be evaluated will be MCAR [12]. Assuming that the data is missing completely at random, **relaxes** the problem of generating missing data, since there is no structure underlying the appearance of a Missing Point. A sample is drawn

from a **Uniform Distribution** $\mathcal{U}$ [12], to determine if we should turn a data point into a Missing Point.

Algorithm 2: Missing Data Induction Algorithm

Result: $\mathcal{M}_{id}$
$MR \leftarrow \frac{MR}{MS}$
$u_N \leftarrow \mathcal{U}.sample(N)$
$\mathcal{M}_{id} \leftarrow (u_N > MR)$
if *MS* $<$ *1* **then**
  $\mathcal{M}_{id} \leftarrow ExtendSequences(\mathcal{M}_{id})$
end

The **Missing Rate** is factored in function of **Missing Sequences**, so that the total number of Missing Points, approximates the desired **rate**. Masks with Missing Sequences will be **extended**, so that the sequential points after the initially selected point, will also be Missing Points.

4.8. Baselines

1. **Mean Imputation** - All the data underwent *standard scaling*, meaning that every variable has zero mean and unit variance. The greediest approach to imputation is to use the mean value.

2. **Forward Filling** - Consists on filling missing data with the last observed value.

3. **Iterative PCA** - An IPCA algorithm [10] was implemented with the help of the main PCA algorithm being developed by Scikit-Learn [3]. The algorithm is initialized with zero value, and to $k = 3$ components. The tolerance for convergence and stopping condition, was set to $\epsilon = 1e - 5$.

4. **K-Nearest Neighbors** - Non-parametric and robust. It has been introduced in the Scikit-Learn package [3] [17], in version 0.23. The neighbourhood was defined to have $k = 7$ neighbors.

5. **Denoising Autoencoder** - As shown in [8], DAE have a **good overall performance** across a variety of Imputation tasks.

6. **Variational Autoencoder** - Deep Generative models have been shown to improve [14] missing data imputation and is a **new outlook** on the old problem that is Missing Data Imputation.

4.9. Training Regime

The model was optimized using a variation of MSE, introduced in the proposal section. The **Adam optimizer** proposed by Kingma et. al [11] is used, with a learning rate used across all experiments of $1e^{-3}$, and a weight **decay factor** of $1e^{-5}$.

The weights are initialized using **Xavier Uniform Initialization** introduced by Xavier and Bengio [7], after showing that Neural Networks might have problems converging from random initialization. The weight is drawn from a Normal distribution with zero mean, and with $\frac{\sqrt{6}}{\sqrt{\#inputs+\#outputs}}$ variance, where $\#$ is the cardinality of a set.

The models were trained for a maximum of 200 epochs, with **Early Stopping**. It is shown that continuing to train a model after it reaches a convergence plateau exacerbates overfitting. [4]. Additionally, we back the Early Stopping condition for the FFTron model, following the works of Yanyang et al., and the F-Principle [13] [21].

The model was trained using batches of size 64. The windows had a empirically determined length of 70 time-steps. For reproduce-ability purposes the random seed generator was set to 7. The model is optimized using complete data, being one of the **drawbacks** of the proposed solution.

Hinton et al. introduced the **Dropout** technique, as means to reduce overfitting [16]. The Dropout in the input layer, synergizes with the missing data algorithm zero initialization. Data is being dropped at random, and it is shown to improve model **robustness** and **stability**, allowing better imputation estimates.

5. Results & discussion

5.1. Imputation Results

In general, every Imputation Method will incur more error as the Rate increases. Interestingly, the inverse tendency is observed when analyzing results in function of Missing Sequence. With exception of Forward Fill, and KNN for the Household-Consumption dataset, all other experimented methods showcase a better performance when the Missing Sequences are larger.

The KNN algorithm shows **dominating performance** across all metrics, including a solid performance in terms of R2 - meaning that the model fit prediction is still good when distant from the mean.

Even though PCA and KNN performs better for the Wind-Turbine dataset, in the Household-Consumption dataset were the **worst performant** methods, corroborating the fact that there is **no clear answer** as to which imputation method one should use. We verify that the performance of both of these algorithms is tied to the correlation of the dataset. If the data is correlated these algorithms will perform well. Wind-Turbines are naturally more correlated than human household electrical-consumption.

Recurrent models like the Seq2Seq (S2S) and LSTM-AE were the first research avenue. These models performance for the imputation task is on average worse than simple statistical imputation. The huge gap of performance between the Recur-

Table 1: Missing Data Imputation results for the Wind-Turbine and Household-Consumption respectively.

Method	MSE	MAE	RMSE	R2
Mean	1.64	1.44	3.18	4.99
FFill	1.31	1.0	2.68	5.5
PCA	0.429	0.527	1.29	6.51
KNN	**0.299**	**0.346**	**1.0**	**6.69**
LAE	1.06	1.1	2.49	5.69
DAE	1.09	1.12	2.53	5.65
VAE	1.59	0.855	1.97	6.14
S2S	1.82	1.5	3.31	4.8
LSTM-AE	1.76	1.49	3.26	4.86
FFTron	0.645	0.805	1.87	6.22

Method	MSE	MAE	RMSE	R2
Mean	1.88	1.15	3.24	4.19
FFill	1.82	**0.792**	3.04	4.29
PCA	3.01	1.56	3.93	3.13
KNN	2.38	1.08	3.5	3.83
LAE	1.55	0.999	2.89	4.53
DAE	**1.53**	0.997	**2.87**	**4.56**
VAE	1.59	1.03	2.94	4.49
S2S	2.01	1.27	**2.87**	3.86
LSTM-AE	2.07	1.27	2.93	3.79
FFTron	1.55	0.988	2.9	4.54

rent Models and a KNN-Impute approach, led this research avenue to a **standstill**, given that KNN already performs pretty well for the case of the Wind-Turbine dataset (sponsored data). The KNN is **costly**, and as so time and resources are critical, the Iterative PCA approach achieved very competitive results for an industrial grade dataset.

Linear Autoencoders (LAE, DAE, VAE) achieved better results on average than the Recurrent Autoencoders (LSTM-AE, S2S) and than simpler imputation methods. The Linear Autoencoders all have very similar R2 metric values in function of Missing Rate, meaning that they perform well when distant from the mean. Though, linear models can't exploit temporal relationships.

Following the better performance of linear models, the FFTron was developed - both temporal and linear. The FFTron model managed a better performance than the Linear Autoencoders in this dataset. Even though the method can't beat the KNN imputation (as well as PCA), it managed a performance score in between the KNN approach and Linear Autoencoder approach.

The FFTron, a more principled approach for handling Time-series data, had a performance similar to other linear methods. Furthermore, this model is not dependent on dataset correlation. We consider, that overall, the FFTron is a better approach than the baselines.

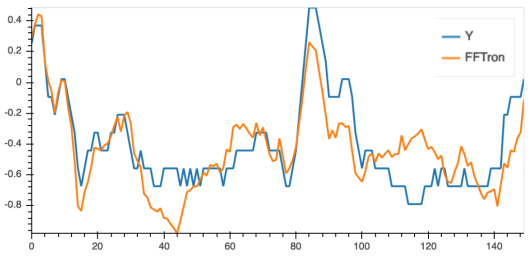

Figure 2: FFTron prediction with heavy latent-space compression - 3 latent variables.

5.2. Latent-Space Analysis

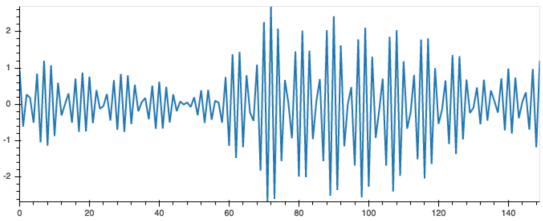

Figure 3: Applying a IDFT on a latent-variable, we get a latent-time signal.

The FFTron latent-space is the heart of the FFTron generalization capabilities. As argued in the motivation of this model architecture, by the **F-Principle**, the model generalization capabilities lie in the dominating low-frequencies.

Additionally, when the latent-space variables (in frequency) are inverted into the time-domain, the signal reveals harmonic structure. This phenomenon occurs if the latent-space is minimally compressed - meaning that the model's latent-space dimension should be at least one dimension smaller.

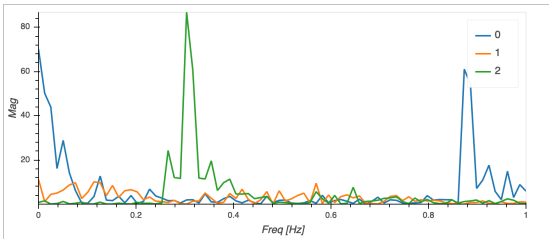

Figure 4: The latent-space spectrum, with 3 latent variables.

The main differences between frequency components of a compressed latent-space and a non-compressed latent-space is the **distribution** of the components, in frequency-space. When the latent-space is compressed, instead of being low-frequency component dominated, the components are **non-overlapping** on their peaks, and **well dis-**

Table 2: Comparison between FFTron model and LSTM-AE model for latent-spaces with 3 dimensions.

Model	FFTron	LSTM-AE	KNN	PCA
Time/Epoch	**0.93**	5.48	-	-
Epochs	**37**	138	-	-
Test Error (Mae)	**0.24**	0.35	-	-
Parameters	**52**	268	-	-
Inference (s)	**0.003**	0.03	-	-
Imputation (s)	**0.13**	0.54	1.99	0.84

tributed through the spectrum.

We also observe that the FFTron doesn't necessarily separate singular frequency components into different latent-variables. We can observe in Fig. 4 that the first latent-space variable (in blue) has two frequency components - a low-frequency and a high-frequency component. While the second latent-space variable (in orange), doesn't seem to have much expressiveness, when compared to the other variables.

Since each of the latent-space variables is a frequency spectrum, we can use a **Spectrogram** to visualize the evolution of the latent-space variables over the test-set. Fig 5 reveals the spectrogram for one latent-variable. Although only empirically ascertained, we observe that the spectrogram is dominated by specific frequencies. The number of dominating frequencies in the spectrogram will equal the number of dimensions in the input-space. This might reveal that the latent-space is in fact storing information on each feature.

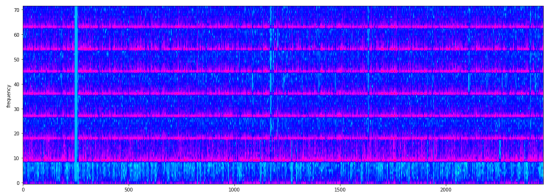

Figure 5: The latent-space spectrum, with 3 latent variables.

5.3. Imputation Method Comparison

The methods are compared using the Household-Consumption dataset. The models were trained using a CPU, in order to assess the usability in low-end hardware. We additionally compare algorithm times for the imputation of 20000 data points.

We observe that the FFTron can get a **better performance** than Recurrent Models. The FFTron model can **converge faster** than a Recurrent Model, due to two reasons: 1) the model has 5 times **less parameters** to optimize, than the structural equivalent with Recurrent Networks; 2) the model is **linear**, which allows faster convergence than non-linear models, like Recurrent Networks.

The time taken per epoch also reveals this 1:5 ratio. It takes on average 5 times **less to train** an FFTron than a Recurrent Model. Inference time is also a huge computational win for the FFTron taking 10 times **less to infer** on a batch of size 4. This means that the FFTron will be on average, a faster than a Recurrent Model to do an imputation. We can observe that the FFTron took 4 times **less time to impute** a dataset of 20000 points. For comparison, the highly-performant industry grade KNN imputation method, takes on average 2 seconds to process 20000 data points and 7 features. We remind the reader that the dimensions of this test dataset does not represent an Industry-Grade dataset, that can easily reach 1 million data points with more than 50 features.

6. Conclusions

We consider that, for the datasets evaluated, the FFTron is the best performant overall in the imputation task.

The FFTron is a **lightweight** model. The training procedure is more **swift** and **cheaper** taking 25 times less time to converge than the Recurrent Network version. The model can output **long predictions** with **low computational cost**, with **less memory** and **runtime** - while being backed by more than a century of research into Fourier Transformations.

Moreover, the model offers **good latent-space capabilities**. We found that the latent-space frequency representations of the FFTron model build a spectrum of dominating frequencies. It was found that by compressing the latent-space the model was forced to forego some information, and started **populating other frequency bands**, besides the low frequencies. The spectrogram of the latent-space shows a number of **dominating frequencies** throughout time, that match the number of input dimensions - irregardless of the latent-space dimensions.

The proposed model achieves **good results** in both. It is not the best candidate, but by a very small margin in all metrics. The error margin comes as a trade-off for a still largely **unexplored type of latent-space**.

We propose the following routes of research as future work. For the imputation procedure using the FFTron:

1. **Optimization Procedure** - The current model still requires to be trained on complete data, which is not always a possibility. Furthermore, the algorithm has to be initialized with zero values, which can introduce **artifacts**, that *pollute* the imputation.

2. **Frequency Space Imputation** - Due to the fact that the FFTron models are based on Fourier Transformations, means the Missing Data problem can be handled using **traditional spectral analysis**. Examples include algorithms in the area of Least Squares Spectral Analysis - interpolation in frequency-space can be done with the *Lomb-Scargle* algorithm.

3. **Increased Sequential Performance when Resampled** - The phenomenon of increased performance with longer Missing Sequences, when the data is resampled, still remains a mystery and was not looked into.

Questions still unanswered on the FFTron latent-space:

1. **Spectrogram Phenomenon** - Why does the latent-space (log) spectrogram showcase a number of dominating frequencies equal in dimension to the input space? Additionally, this phenomenon is not fact, and needs to be verified with more data;

2. **Latent-Space Embeddings** - The richness of the found latent-space representations is left **unexplored**. We recommend exploring if similar representations in latent-space translate into similar input windows. If this is the case, the imputation method can be extended by using similar signals from the training set to impute Missing Values.

3. **Harmonic Distribution** - Although not mentioned in the results, we suspect that when the latent-space variables contain only one frequency component, it shows a similar shape of a Harmonic Distribution. It is recommended to look into Probabilistic Programming.

References

[1] Ieee standard for binary floating-point arithmetic. *ANSI/IEEE Std 754-1985*, pages 1–20, 1985.

[2] N. Balakrishnan, T. Colton, B. Everitt, W. Piegorsch, F. Ruggeri, and J. L. Teugels, editors. *Wiley StatsRef: Statistics Reference Online*. Wiley, Apr. 2014.

[3] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[4] R. Caruana, S. Lawrence, and C. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. volume 13, pages 402–408, 01 2000.

[5] D. Dua and C. Graff. UCI machine learning repository, 2017.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representation by Error Propagation*, volume Vol. 1. 01 1986.

[7] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. JMLR Workshop and Conference Proceedings.

[8] L. Gondara and K. Wang. Mida: Multiple imputation using denoising autoencoders, 2017.

[9] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[10] H. A. L. Kiers. Weighted least squares fitting using ordinary least squares algorithms. *Psychometrika*, 62(2):251–266, Jun 1997.

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

[12] R. J. A. Little. *Statistical Analysis with Missing Data*. Wiley-Interscience, sep 2002.

[13] T. Luo, Z. Ma, Z.-Q. J. Xu, and Y. Zhang. Theory of the frequency principle for general deep neural networks, 2019.

[14] P.-A. Mattei and J. Frellsen. Miwae: Deep generative modelling and imputation of incomplete data, 2018.

[15] J. O. Smith. *Spectral Audio Signal Processing*. `http`:http://ccrma.stanford.edu/ jos/sasp///-`ccrma.stanford.edu/~jos/sasp/`, 2020. online book, 2011 edition.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from

overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 06 2014.

[17] O. Troyanskaya, M. Cantor, G. Sherlock, T. Hastie, R. Tibshirani, D. Botstein, and R. Altman. Missing value estimation methods for dna microarrays. *Bioinformatics*, 17:520–525, 07 2001.

[18] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. *Universit e de Montr eal, Dept. IRO, CP 6128, Succ. Centre-Ville, Montral, Qubec, H3C 3J7, Canada*, 2008.

[19] A. Wichert. *Intelligent Big Multimedia Databases, World Scientific*. 07 2015.

[20] M. Wolter, J. Gall, and A. Yao. Sequence prediction using spectral rnns, 2018.

[21] Z.-Q. J. Xu, Y. Zhang, and Y. Xiao. Training behavior of deep neural network in frequency domain, 2018.