Erasmus University Rotterdam
Erasmus School of Economics
Bachelor Thesis International Bachelor Econometrics and Operations Research

Learning Missingness Mechanisms for Imputation with Denoising Autoencoders

Henry Karl Bernhard Heppe (597651)

Supervisor:	Markus Müller
Second assessor:	Maria Grith
Date final version:	28th June 2024

Abstract

Deep learning-based models for missing value imputation can often outperform well-known statistical imputation methods such as regression-based imputation. Strong results can be achieved by using Denoising Autoencoders (DAE). These are neural networks that learn to reconstruct an observation from a corrupted version of itself. They effectively learn to reverse the corruption process that is artificially applied to the observation. Previous works use DAE for imputation with a manually specified corruption process. In this study, we propose the *imputeLM* imputation method. It is based on the idea of learning a model on the missingness mechanism itself. This model can then produce corruptions for the training of the DAE, which are more similar to the missingness patterns that one is imputing. Therefore, the DAE learns to reverse a closer representation of the missingness mechanism, which results in a theoretically more accurate imputation. We evaluate this hypothesis across different missingness scenarios for the MNIST, FashionMNIST and CIFAR10 datasets and find that incorporating learned missingness in the imputation process can be beneficial for difficult imputation tasks. Additionally, we also show that conditioning the corruption process on higher-level representations of the data can help generate more realistic missingness patterns.

1 Introduction

When working with real-world data, one of the first challenges on the way to a functioning model is dealing with missing values. In most data collection scenarios, certain variables may not be observed for the whole sample. This can be due to humans making errors and forgetting to fill in a value, it can be survey participants dropping out of the survey but it can also arise naturally from the context. Missing values are not only commonly encountered in tabular data but are also a problem for image data. Whether the task is repairing a corrupted image, denoising a bad image or removing an element of the image and doing inpainting on the area, they can all be cast as missing value imputation problems.

In the statistical literature, the mechanisms that give rise to missing data are grouped into three categories. Depending on whether the missingness depends on both observed and unobserved, only observed or no variables at all the mechanisms are classified as *Missing Not at Random* (MNAR), *Missing at Random* (MAR) or *Missing Completely at Random* (MCAR) respectively (Little and Rubin, 2019).

A common approach to imputation is to frame the imputation task as a repeated supervised learning problem where one observed variable is the dependent variable and all others are the independent variables (Van Buuren and Oudshoorn, 1999). Then, the learned model is used to impute the missing values of the dependent variable. Since this requires estimating one separate model per variable, this approach becomes infeasible quickly for high-dimensional data, large numbers of observations or more complex models. An approach that has gained popularity in recent years is imputation with deep learning-based generative models (Yoon et al., 2018). One such model architecture that lends itself naturally to data imputation is the Denoising Autoencoder (DAE) introduced by Vincent et al. (2008). These are neural networks that are trained to reconstruct an observation from a corrupted version of itself. Their reconstructive training enables them to reconstruct missing values just as well (Pereira et al., 2020).

Previous literature on using DAE for imputation has two caveats. Yoon et al. (2018) find that many of them require complete data with missingness masks also provided to have a learning task where the ground truth is known. In practice, this is usually not the way the missing value problem presents itself. The second caveat they find applies if the dataset can be split into observations that are fully observed and observations that are partially observed. Most DAE-based models train only on the fully-observed observations with a manually specified corruption process. This has the disadvantage that it leaves out the information from the latter part of the dataset completely, i.e. the information about how the observed missingness pattern is connected with the variables themselves. One can conjecture that including information about the missingness mechanism in the imputation model might aid imputation performance. Ma et al. (2020) take a step in that direction in that they distinguish three simple missingness patterns to which they manually design different imputation mechanisms.

The approach proposed in this study tackles both caveats by taking the idea of adapting the imputation model to the missingness mechanism further. It leverages the observations that contain missing values to learn a model on the missingness patterns. This model is then applied to the completely observed samples as a corruption process to obtain a training set for the DAE. Since a DAE effectively learns to invert its corruption process (Bengio et al., 2013), it

implicitly learns to invert the learned missingness mechanism and impute the missing values well. Being trained on data that is more similar to the actual missing data, this DAE is then well-suited for the specific imputation task. Due to the *learned missingness* component, the model is called *imputeLM*[1]. To make the learning of the missingness more robust against the missingness itself we propose to condition this learning on a dense data representation without missingness instead of the actual observations. These representations are obtained from an Encoder as part of a separate DAE trained on a standard corruption process. On a high level, the idea of the *imputeLM* model is that by making use of all the available information and adjusting the training of a DAE flexibly, it can perform better at imputing. We verify the imputation performance of the model on the three well-known datasets MNIST (LeCun, 1998), FashionMNIST (Xiao et al., 2017) and CIFAR10 (Krizhevsky and Hinton, 2009) for a range of simulated missingness scenarios including MAR, MNAR, MCAR and inpainting tasks.

The contributions of this study can be summarized as follows:

1. For missing value imputation, we propose the *imputeLM* model: a modified Denoising Autoencoder, for which the corruption process is learned on real missingness.

2. We show how missingness patterns can be learned on partially observed data to generate similar artificial missing values for fully observed data.

3. We show that conditioning on higher-level representations of the data obtained from a separate Encoder model can yield better generalization performance for the missingness generating model.

4. The presented numerical results indicate that our method can outperform comparable models significantly across multiple scenarios, datasets and missingness levels.

The remainder of this thesis is structured as follows. Section 2 provides an overview over the preceding literature. Section 3 introduces the missing value theory more thoroughly and defines the proposed method. In Section 4, we provide empirical results comparing the method to existing ones, and Section 5 concludes the paper.

2 Literature Review

The foundation for the statistical analysis of missing data mechanisms and subsequent building of imputation models is laid by Rubin (1976). He derives a set of general weak conditions under which ignoring the mechanism behind the missing data is acceptable for statistical inference. From these conditions, a taxonomy of three classes of missingness algorithms has emerged. The processes are missing completely at random (MCAR), missing at random (MAR) and missing not at random (MNAR). MCAR entails that the missingness does not depend on any observed or unobserved variable, while for MAR, it depends on observed variables and for MNAR, it depends on unobserved and possibly observed variables.

[1] A Python implementation is provided at https://github.com/henry-heppe/imputation_with_learned_missingness.

Little and Rubin (2019) identify four categories of approaches dealing with missing data. The first is the idea that one simply discards incomplete observations - this is called complete-case analysis and has the natural advantage that it is easy to implement. They find that its problem is that it works only for limited amounts of missing data and can lead to severely biased inference. The second category are weighting procedures. These follow the idea of reweighting the observed sample to adjust for the effects of the non-observed data points - an approach that they find mainly applies to survey-based data. The third method are model-based approaches for which a model of the complete data is specified, and inference is based on this model via maximum likelihood. Little and Rubin (2019) mention that this approach has the advantage that it is flexible and makes the underlying assumptions explicit so they can be evaluated. A fourth category of methods is imputation, where the missing values are replaced by a sensible estimate. This estimate can be anything from a simple mean to a regression-based or more complex estimate. To incorporate the uncertainty in the estimate of the missing value, simple imputation can be extended to multiple imputation, where more than one estimate of the missing value is drawn from the predictive distribution and then combined into one value.

The method proposed in this study is an imputation method. It is by default a method for single imputation but has a simple extension to multiple imputation if a generative model is chosen accordingly.

Within the imputation methods, a common distinction is made between discriminative and generative models (Yoon et al., 2018). A few well-known discriminative methods are MICE (Van Buuren and Oudshoorn, 1999), missForest (Stekhoven and Bühlmann, 2012) and softImpute (Mazumder et al., 2010). However, in cases of high-dimensional data, these methods can become computationally infeasible. In this study, we thus focus on generative models. These are most often based on deep learning architectures, two subtypes of which we consider here: Denoising Autoencoders (DAE) and Variational Autoencoders (VAE). Pereira et al. (2020) find in their survey that these often outperform classical statistical methods on (tabular) data imputation tasks. DAE (Vincent et al., 2008) are trained by reconstructing an observation from a corrupted version of itself which implies that the application to imputation comes naturally. Bengio et al. (2013) describe how a DAE learns the reverse conditional distribution of its corruption process. Thus a central element in the construction of these models is the choice of the corruption mechanism. Commonly considered variants are setting features randomly to zero or adding Gaussian noise. Ma et al. (2020) use DAE for imputation with a corruption process that is manually adapted to the missingness patterns. They propose special corruption processes for univariate and monotone missingness, which are employed after manually identifying such a mechanism. The approach proposed here relates to their idea of adapting the corruption process. However, instead of manually identifying and adjusting it, our method tries to learn the optimal representation of the true missingness mechanism directly.

Variational Autoencoders learn to encode the observation to a latent probability distribution (Kingma and Welling, 2013) and subsequently reconstruct the observation from a sample of that distribution. This has the advantage that by manually specifying and varying the parameters of the latent distribution and then applying the decoder, a controlled generation of new data points is possible. VAE have enjoyed substantial popularity for imputation in recent years

(Boquet et al., 2020), (McCoy et al., 2018) and are included here as a benchmark model.

While some of the previous works describe exact models to use for imputation, our method can be interpreted as a model framework consisting of learning the missingness and then learning to replace the missingness. This framework can be filled with a wide range of model architectures fitted to the task at hand.

3 Methodology

In this section, a formal definition of the missing value problem is given. It is followed by an overview of the *imputeLM* framework proposed here, after which the artificial neural networks employed in this study are described.

3.1 Missing Values

3.1.1 Missing Value Theory

We formalize the missing data problem with notation based on Ipsen et al. (2020). We denote the data matrix $X = (\mathbf{x}_1, ..., \mathbf{x}_n)^T \in \mathcal{X}^n$, assuming that each row is one of n i.i.d realizations of the random variable $\mathbf{x} \in \mathcal{X}$ where $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_p$ is our p-dimensional feature space. We denote the i-th observation for the j-th variable as x_{ij}. The set of indices of all observations consists of two disjoint subsets $\mathcal{I} = \mathcal{I}_m \cup \mathcal{I}_o$ where the former describes the observations for which at least one variable is not observed. We denote the respective number of observations as n_m and n_o. If observation i has missing values, we write $\mathbf{x}_i = (\mathbf{x}_i^o, \mathbf{x}_i^m)$ to differentiate the observed features from the missing ones. To define the missingness process itself, we use the masking matrix $S = (\mathbf{s}_1, ..., \mathbf{s}_n)^T \in \{0, 1\}^{n \times p}$ where $s_{ij} = 1$ if x_{ij} is observed and 0 if it is missing. In our notation, we distinguish the underlying true variables $\mathbf{x}$ from the observed vector $\tilde{\mathbf{x}}$ which has *missing* values where $s_{ij} = 0$. The joint probability distribution of the underlying $\mathbf{x}$ and $\mathbf{s}$ parameterized with γ and θ is

$$p_{\gamma,\theta}(\mathbf{x}, \mathbf{s}) = p_\gamma(\mathbf{x}) p_\theta(\mathbf{s}|\mathbf{x}). \tag{3.1.1}$$

Ipsen et al. (2020) then define the three missingness mechanisms in terms of the conditional distribution of $p_\theta(\mathbf{s}|\mathbf{x}^o, \mathbf{x}^m)$. Missing completely at random (MCAR) means that $p_\theta(\mathbf{s}|\mathbf{x}) = p_\theta(\mathbf{s})$, i.e. the probability of a value missing is independent of any observed or unobserved variables. In the missing at random (MAR) scenario $p_\theta(\mathbf{s}|\mathbf{x}) = p_\theta(\mathbf{s}|\mathbf{x}^o)$ thus the missingness depends only on observed variables. For the case of missing not at random (MNAR), it holds that $p_\theta(\mathbf{s}|\mathbf{x}) = p_\theta(\mathbf{s}|\mathbf{x}^o, \mathbf{x}^m)$ depending on both observed and unobserved variables.

To facilitate the explanation of how the corruption process in a Denoising Autoencoder ties in with this inference task later on, we slightly abuse the notation of Ipsen et al. (2020) to frame the imputation problem in terms of estimating $p(\mathbf{x}|\tilde{\mathbf{x}})$, i.e. finding the distribution of the complete data given knowledge of the data with missing values. The possibility of finding a precise estimate for the underlying data distribution then still depends on our ability to model the missingness process $p_\theta(\mathbf{s}|\mathbf{x})$ as $\tilde{\mathbf{x}}$ depends on $\mathbf{s}$ and $\mathbf{x}$.

3.1.2 Simulating Missingness

To examine the performance of the proposed model, we simulate several missingness-generating processes on originally complete datasets. The first three are implementations of a general type of MCAR, MAR and MNAR processes. For the MCAR process, we generate the missingness from a standard uniform distribution $\mathcal{U}(0,1)$ and then adjust for the specified missingness proportion per observation. The general MAR and MNAR processes are based on a formulation by Yoon et al. (2018). Let the probability that observation x_{ij} is missing be denoted by p_{ij}. For the MAR process, they define

$$p_{ij} = \frac{p_j \cdot n \cdot \exp(-\Psi(i,j))}{\sum_{l=1}^{n} \exp(-\Psi(l,j))}, \tag{3.1.2}$$

where

$$\Psi(i,j) = \sum_{k<j} \left(w_k s_{ik} x_{ik} + b_k(1 - s_{ik})\right). \tag{3.1.3}$$

Here, p_j is a fixed missingness proportion chosen for variable j and $w_k, b_k, \forall k = 1, \ldots, p$ are weights and biases sampled from $\mathcal{U}(0,1)^p$ once for the whole missingness process. s_{ij} is the missingness indicator as defined above. The missingness mask is then generated for one variable after the other taking into account the feature values and missingness values of all previous variables. To generate a mask from the probabilities, s_{ij} is sampled from the Bernoulli distribution $\mathcal{B}(p_{ij})$. Visualizations of the missingness mechanisms are included in Section 4.4 and Appendix A.

Because of the high dimensionality of the datasets used in the experiments in this study, it is not feasible to specify the missingness proportions p_j by hand. Instead, we sample them from a multivariate normal distribution with mean zero and as covariance matrix the empirical covariance matrix of the data. To make the matrix positive definite, a fraction of the identity matrix is added to it. The samples are then passed through a sigmoid function to bring them into $[0,1]$ range. With line search on the offset term of the sigmoid function, we control the amount of missingness introduced by this process. We choose the multivariate normal instead of, for example, a uniform distribution because it introduces an extra layer of structure depending on the real data. Using the uniform distribution instead would make this MAR generation more similar to the MCAR process and, thus, a less interesting task for learning information from the process. It is worth noting that this way of sampling the missingness proportions also introduces a small amount of interdependence between the simulated missingness and potentially unobserved variables. This does not diminish the utility of this simulation, though, as it theoretically only makes the imputation task more difficult and creates a more realistic scenario.

The above formulation from Yoon et al. (2018) can be implemented efficiently by making use of optimized matrix multiplication. For this, we rewrite the formulation as a column generation algorithm to reduce the number of duplicate operations. We compute Ψ_j (a vector with $\Psi(1,j), \ldots, \Psi(n,j)$ as elements) as

$$\Psi_j = \operatorname{diag}(A_{1:j-2} S_{1:j-2}^T) + \mathbf{b}_{1:j-2} S_{1:j-2}^{*T} + \operatorname{diag}(A_{j-1} S_{j-1}^T) + \mathbf{b}_{j-1} S_{j-1}^{*T}). \tag{3.1.4}$$

Here, $A = XW$ is a $n \times p$ matrix composed of the data matrix X and the weights matrix W, which is a diagonal matrix with the weights w_j on its diagonal. The matrix subscripts refer to the subset of columns selected from the respective matrix, i.e. $1 : j-2$ selects the first $j-2$ columns. The operation $\text{diag}(\cdot)$ extracts the diagonal elements of a matrix as a column vector. S is the mask matrix as above and $\mathbf{b}$ a row vector of the biases b_j. The matrix S^* is defined as $s^*_{ij} = 1 - s_{ij}$. The complete matrix A only needs to be calculated once in the beginning. For every j, all components that are subscripted with $j-2$ are already known from beforehand, and all with $j-1$ are newly available from the previous iteration. Thus only the last two matrix multiplications need to be performed when computing the next Ψ_j. This vector is then transformed according to Equation 3.1.2 with the exponential applied element-wise. The derivation of this formulation can be found in Appendix B. Since this process does not guarantee the resulting p_{ij} to be less than one, we clip any values larger than one. Additionally, in practice, the value of Ψ_j may accumulate to such large numbers that the denominator becomes zero, and the probabilities are not a number. In this case, we reset Ψ_j to zero.

The MNAR process is simulated similarly to MAR (also due to Yoon et al. (2018)). The p_{ij} are defined as in Equation 3.1.2 but depending only on x_{ij} through

$$\Psi(i, j) = w_j x_{ij}. \tag{3.1.5}$$

The matrix formulation of this process then simplifies to plugging in columns of A into Equation 3.1.2 one after the other.

In this study, we also consider another type of MNAR scenario which is based on masking quantiles taken from Muzellec et al. (2020). This mechanism first randomly splits variables in X into $\mathbf{x}^o$ and $\mathbf{x}^m$ based on a prespecified proportion. It will then set the top and bottom q-quantiles for these variables to be masked such that a specified share of observations is covered. Because the quantile information is with respect to the variable that is masked, this is an MNAR mechanism, which we call QMNAR in this study to distinguish it from the MNAR introduced above.

While these mechanisms can apply to any numerical dataset, we also include two types of missingness that are more specific to image data and provide an even more explicit structure to be learned. The first one we refer to as PATCH, which consists of masking one contiguous patch of the image from left to right. The height of this patch is determined by the level of missingness, and the vertical position is uniformly random. The second type, PATCHES, are multiple small quadratic patches (e.g., 5 by 5 pixels) placed randomly on the image, where the number of patches is determined by the overall missingness level. Placing these processes in one of the three missingness assumptions is less straightforward since the missingness does not depend on the value of any of the variables. However, it is a special type of MCAR because the missingness values depend on each other through the spatial correlation of pixels.

3.2 imputeLM Framework

In this section, we propose the *imputeLM* model for missing value imputation and elaborate on the three separate neural networks that this model is made up of in Section 3.3. These three

neural networks are trained sequentially, but during inference, the actual imputation is only done using the last model. The complete model consists of a Preprocessing Denoising Autoencoder (PDAE) of which only the encoder part is used followed by a Missingness Predictor (MP) model which in turn is followed by another Denoising Autoencoder, which we here refer to as the Adapted Denoising Autoencoder (ADAE) to distinguish it from the PDAE. Figure 3.2.1 illustrates the training of the ADAE model. A visualization of all training steps can be found in Appendix C.

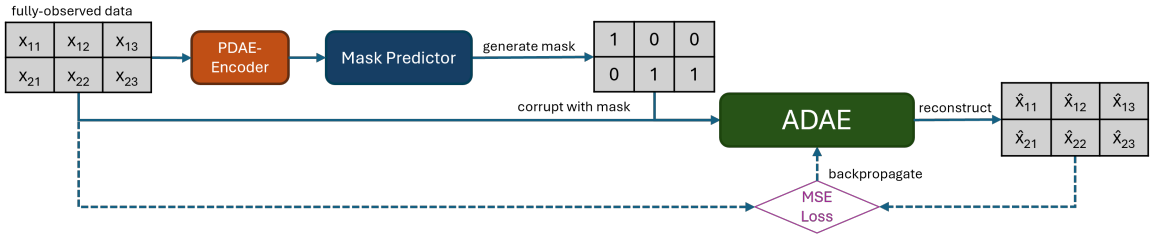

Figure 3.2.1: Training setup of the Adapted Denoising Autoencoder, which is used to impute the missing values. See Appendix C for full model visualization.

We discuss the theory of these models in reverse order with respect to the data flow since, in this order, the shortcomings of one model justify the existence of the next model.

3.2.1 Adapted Denoising Autoencoder

The fundamental concept of our proposed method is that of an autoencoder. An autoencoder is a model that consists of an encoder function $f_\delta(\mathbf{x})$ and a decoder function $g_\omega(\mathbf{x})$ (Vincent et al., 2008). The encoder defines a deterministic mapping from an input $\mathbf{x}$ to a latent representation $\mathbf{z}$. The decoder defines another mapping from $\mathbf{z}$ to a reconstruction of $\mathbf{x}$ called $\hat{\mathbf{x}}$. Together the encoder and decoder form a model that takes as its input some observation $\mathbf{x}$ and has as its target that same input. Thus it does not require pairs of data and labels but is trained in a self-supervised manner only.

One of the original purposes of autoencoders is to learn useful latent representations of the input which can be used as inputs to other models. However, in the setup presented until now, the model has no incentive to learn anything apart from the identify function since the input and the target are the same.

The Denoising Autoencoder (DAE) introduced by Vincent et al. (2008) solves that problem. It does so by training the model to reconstruct $\mathbf{x}$ from a corrupted version $\mathbf{x}^*$ of itself. Adjusting their notation slightly, we can define a DAE in terms of the optimization problem minimizing a loss between the true $\mathbf{x}$ and its reconstruction

$$\hat{\delta}, \hat{\omega} = \arg\min_{\delta,\omega} \frac{1}{n_o} \sum_{i \in \mathcal{I}_o} \mathcal{L}\left(\mathbf{x}_i, g_\omega(f_\delta(\mathbf{x}_i^*))\right). \tag{3.2.1}$$

While this equation applies to any DAE over a general dataset, we specify the dataset of fully-observed samples here to emphasize which part of the dataset the model of this section is trained on.

A corruption process used for training a DAE can, for example, be setting a number of the

elements of $\mathbf{x}$ to zero. The model then learns to infer the removed information from the other variables. This forces it to learn higher-level representations of the data that are invariant to such small corruptions (Vincent et al., 2008). To the application of missing value imputation, the original goal of learning representations is not central, but the very learning setup - learning to fill in missing information in the input data - lends itself naturally.

To formalize the connection between DAE and imputation, we define the corruption process to be a stochastic mapping $\mathbf{x}^* \sim p(\mathbf{x}^*|\mathbf{x})$. In words, it defines the conditional distribution of the corrupted input given the real input. When this stochastic mapping is used to generate pairs $(\mathbf{x}^*, \mathbf{x})$ that are used to train a DAE, the DAE effectively learns to reverse this process (Bengio et al., 2013). This means it estimates the reverse conditional distribution $p(\mathbf{x}|\mathbf{x}^*)$. To then optimize the DAE for an imputation task we thus set $\mathbf{x}^* = \tilde{\mathbf{x}}$ and train a DAE on a corruption process $p(\tilde{\mathbf{x}}|\mathbf{x})$ of which the reversal is the actual imputation process $p(\mathbf{x}|\tilde{\mathbf{x}})$. The reversal learning property implies that training the DAE on samples generated by the true stochastic missingness mechanism would yield a consistent estimator of the true imputation function $p(\mathbf{x}|\tilde{\mathbf{x}})$.

If, in this setup, we use one of the common corruption processes, such as setting elements randomly to zero, one can see that its interpretation as a missingness mechanism would classify it as MCAR. This is because, for each variable, the decision of whether or not it is corrupted is independent of all variables in $\mathbf{x}$. Here, the complete corruption process (that is, the conditional distribution of $\tilde{\mathbf{x}}$) is only conditioned on $\mathbf{x}$ with respect to the values of the variables that are not corrupted but not through the mask. We can take this as a theoretical indication that a vanilla DAE should perform similarly to a DAE trained on the true missingness mechanism as corruption mechanism only if the missingness mechanism is MCAR and not otherwise (see Table 4.5.1 for the corresponding experiment). The idea is thus that adapting the corruption mechanism towards the missingness mechanism could provide better imputation performance.

However, while this theoretical consideration is a step towards a more flexible imputation model, in practice, the true missingness mechanism $p(\tilde{\mathbf{x}}|\mathbf{x})$ is unknown. But the partially observed samples $\tilde{\mathbf{x}}_i, \forall i \in \mathcal{I}_m$ can be seen as draws from this distribution. We can thus use these observations to approximate this mapping. It is the core idea of the *imputeLM* model to specify a separate model to learn the missingness mechanism and take on the role of the corruption process for the training of the DAE imputation model. Since the task of learning $p(\tilde{\mathbf{x}}|\mathbf{x})$ is equivalent to learning $p_\theta(\mathbf{s}|\mathbf{x})$, we call this model the Missingness Predictor model and define it in the next section. The ADAE is then *adapted* in that it is trained on a corruption process that adapts to the missingness process differentiating it from the DAE explained in Section 3.2.3.

3.2.2 Missingness Predictor Model

The Missingness Predictor (MP) model is trained to approximate the true missingness mechanism $p_\theta(\mathbf{s}|\mathbf{x})$. Since $\mathbf{x}$ has potentially unobserved components in general, the closest approximation based on observable data is to estimate $p_\theta(\mathbf{s}|\tilde{\mathbf{x}})$. To learn this, the model takes as input a partially observed sample $\tilde{\mathbf{x}}_i, i \in \mathcal{I}_m$ and predicts the corresponding missingness mask $\mathbf{s}_i$. The goal is to use the trained model to replicate the missingness behavior on the fully observed samples to have the ADAE imputation model learn the correct imputation function. If we define

the MP model to be a function $h_\lambda(\mathbf{x})$, we estimate the parameters λ by minimizing a loss as in

$$\hat{\lambda} = \arg\min_{\lambda} \frac{1}{n_m} \sum_{i \in \mathcal{I}_m} \mathcal{L}\left(\mathbf{s}_i, h_\lambda(\tilde{\mathbf{x}}_i)\right). \tag{3.2.2}$$

We can then apply the trained model to the fully-observed $\mathbf{x}_i$ to generate new artificial missingness masks $\mathbf{s}_i$. It is worth noting that $\tilde{\mathbf{x}}_i = \mathbf{x}_i, \forall i \in \mathcal{I}_o$.

To be able to generate masks, our implementation of $h_\lambda(\mathbf{x})$ contains a sigmoid function as the last component, which results in outputs in the $[0, 1]$ range. The output for each variable can thus be interpreted as the probability that that variable is observed. We can compute the loss between these probabilities and the target mask using the binary cross-entropy loss (see Section 3.3). When generating new masks for the training of the ADAE, we turn the probabilities into binary masks by sampling from Bernoulli distributions with the probabilities as their respective parameters. This makes the corruption process an actual stochastic mapping since $h_\lambda(\mathbf{x})$ itself is a deterministic function.

This missingness learning approach has one limitation similar to any other imputation model (see also Section 4.2 and Appendix D). We defined previously that $\tilde{\mathbf{x}}$ is obtained by applying $\mathbf{s}$ to $\mathbf{x} = (\mathbf{x}^o, \mathbf{x}^m)$. If we then also make the distinction between observed and unobserved variables in the realized vector $\tilde{\mathbf{x}} = (\tilde{\mathbf{x}}^o, \tilde{\mathbf{x}}^m)$ we can see in which missingness scenarios the approximation of $p(\mathbf{s}|\mathbf{x}^o, \mathbf{x}^m)$ by $p(\mathbf{s}|\tilde{\mathbf{x}}^o, \tilde{\mathbf{x}}^m)$ is theoretically feasible.

If MCAR is assumed, we know that the missingness mask is independent of all $\mathbf{x}$, and thus, the observed masks are realizations of $p(\mathbf{s})$. The MP model should then learn to regard its input $\tilde{\mathbf{x}}$ as random noise and shape this noise into the distribution $p(\mathbf{s})$. This can still be a meaningful learning task because this distribution can diverge substantially from a vanilla DAE's uniformly sampled masking noise (e.g. if the missingness is PATCH). If one assumes MCAR, one could argue that it could be even better to only give the binary mask as input to the MP model, i.e. turn it into an autoencoder. One could then make it a Variational Autoencoder (Kingma and Welling, 2013) to generate new instances of a learned estimate of $p(\mathbf{s})$.

However, the approach we propose here is supposed to also work in the case of MAR. Because, in this case, the underlying missingness generating mechanism is $p_\theta(\mathbf{s}|\mathbf{x}_o)$, our model of this process needs to be conditioned on the information in $\mathbf{x}_o$ as well. We therefore need to use the partially observed samples $\tilde{\mathbf{x}}$ as input. We can then see that in estimating $p(\mathbf{s}|\tilde{\mathbf{x}}^o, \tilde{\mathbf{x}}^m)$ the model in fact estimates the correct underlying conditional because $\tilde{\mathbf{x}}^o$ is by definition equal to $\mathbf{x}^o$ and $\tilde{\mathbf{x}}^m$ does not have any effect on $\mathbf{s}$.

Only in the MNAR case, we cannot learn $p_\theta(s|\mathbf{x}_o, \mathbf{x}_m)$ properly since $\mathbf{x}_m$ contains relevant information that is not available in $\tilde{\mathbf{x}}^m$. Still, the idea of this model setup is that it learns as much as possible about the distribution from the available information. The MP model should be able to determine the dependence of $\mathbf{s}$ on $\mathbf{x}_o$ through $\tilde{\mathbf{x}}$.

At this point, one more problem appears in the definition of the MP model. Since its main use is to generate new missingness masks for the fully observed samples, but it is trained solely on the partially-observed samples, we need it to have strong generalization capabilities in the direction of partially- to fully-observed. However, to be able to pass a partially observed sample to the MP model as input, the missing values cannot simply be *missing*. A simple solution to

this is to replace the missing values with a fixed placeholder value $a \in \mathbb{R}$ as in Mattei and Frellsen (2019), such that $x_{ij} = x_{ij}$ if $s_{ij} = 1$ and $x_{ij} = a$ otherwise. Then, however, depending on the structure of the data, different values of a would convey different amounts of information about the missingness of a certain x_{ij}. For example, for the MNIST dataset (LeCun, 1998) where the majority of pixels have values close to 0 or 1, replacing missing values with 0.5 would tell the model to simply look for values of 0.5 when trying to infer what values were missing. While the MP model might then be able to achieve near-perfect missingness prediction accuracy during training on the partially-observed samples, this would pose a problem for its generalization ability to the fully observed examples. Since in the actual MNIST images, there are almost no values close to 0.5, a model overfitting on this value generates out-of-sample masks that do not resemble the masks it was trained on. In our theoretical framework, this would mean that in the MAR case, the model would not learn to discard the variables $\tilde{\mathbf{x}}^m$ but rather learn to identify them as missing variables based on their replacement value and use that information to predict their missingness.

A better solution could be to use random noise as a placeholder, i.e. $a \sim \mathcal{U}(0, 1)$. The idea would then be that the model learns that the values used as replacement are not related to the values of the other variables and, thus, must be missing. It could then deduce missingness from the relation of the observed variables. Still, this leaves doubts about the generalizability of its predictions since the partially-observed samples with noise replacement could still be substantially different from the fully-observed samples.

What is needed to theoretically ensure generalizability would be to have as the MP model input a representation of the underlying observation that is (somewhat) invariant to whether the value of a variable is the true one or a replacement of any kind. Ma et al. (2018) propose a *permutation-invariant* encoder based on a Partial VAE to achieve this. In a similar spirit, we propose to use an Encoder obtained from training another Denoising Autoencoder for this task. Recalling that a DAE can be trained to learn higher-level representations of the input invariant to corruptions, such as masking noise, makes it a useful model to be employed in front of the MP model.

3.2.3 PDAE-Encoder Model

To be able to learn MAR missingness patterns the MP model needs to be conditioned on the observed information of a sample. All the inputs to the MP model should be vectors of the same length and cannot have values missing. Thus, we replace the missing values with a placeholder value. The problem is then that the samples with replacement values are different from the fully-observed samples impeding the ability of the MP model to generate masks for the latter while being trained on the former. We, therefore, need to pass the MP model a representation of the observation that is ideally invariant to whether the observation had missing values or not. Thus a model is required that can encode both types of observations into a similar dense representation that does not depend on the missing values. A Denoising Autoencoder is suitable for this since the corruption process used in its training removes the information of some variables and replaces them with a placeholder value. Because this DAE will be applied before the MP model, we refer to it as the Preprocessing DAE (PDAE). For the PDAE to be able to reconstruct

the original observation from the corrupted version, its encoder function produces higher-level representations that are as invariant to this corruption as possible. Because the difference between partially- and fully-observed samples is mainly in terms of the replacement values, the encoder function of the PDAE produces such representations that are (ideally) invariant to whether the observation had missing values or not. When we apply this encoder function to our partially-observed samples with replacement values, the idea is that they are mapped to dense representations without missing values, which we can use as input to the MP model.

We make use of this characteristic by training the PDAE model on the fully-observed samples $\mathbf{x}_i, \forall i \in \mathcal{I}_o$. We denote its encoder by the function $l_\eta(\mathbf{x})$ and the decoder by $m_\phi(\mathbf{x})$. The corruption process for this DAE model is defined by replacing a prespecified proportion of randomly selected variables with draws from $\mathcal{U}(0, 1)$ (Section 4.5 includes an experiment where zero is used as a replacement instead). The proportion of corrupted variables is set to be the proportion of variables that are missing averaged over all observations. Using only the encoder part of the fully trained PDAE as our PDAE-Encoder model, we can then encode the partially observed samples $\tilde{\mathbf{x}}_i, \forall i \in \mathcal{I}_m$ and obtain a dense representation without missing values which we can condition the MP model on. To obtain these representations, the corruption process is not applied.

One remark worth making is that although alleviating the problem of out-of-sample mask generation slightly with the PDAE-Encoder model, it moves the weak point to another part of the model, namely the corruption process used to train the PDAE-Encoder. The more this corruption process diverges from the actual missingness mechanism, the less invariant its representations are to the missingness and its replacement values. However, if one tries to start and adapt the corruption process again towards the missingness one creates a circular problem of learning the missingness but needing representations that are robust to it in the first place. We, therefore, leave it at this architecture of stacking one PDAE-Encoder with one MP model and one ADAE.

With this model, we have introduced the three models that comprise the *imputeLM* model. In practice, they are trained as follows. We first train the PDAE-Encoder on the fully observed samples. We then use it to encode the partially-observed samples, on which we train the MP model. Then we generate new missingness masks conditioned on the encoded fully-observed samples and apply it to the unencoded fully-observed samples with which we train the ADAE. Finally, we do the actual imputation by applying the ADAE model to the unencoded partially observed samples. Appendix E contains a detailed pseudo-code description of the training. In Appendix F, we describe an alternative model setup for the MP model in terms of a Generative Adversarial Network (GAN).

Finally, it is worth noting that the general approach of using a DAE for imputation in this scenario - be it a vanilla DAE or ADAE - relies heavily on the assumption that the observations in $\mathcal{I}_o$ are representative enough of the observations in $\mathcal{I}_m$ to be able to derive meaningful imputation values from the feature relationships found in the former set of observations.

While the explanation of the *imputeLM* setup is kept general on purpose, in the next section, we define the family of functions and their parameterization that we use for each of these models.

3.3 Fully-connected Neural Networks

In this study, all of the previously mentioned models consist of fully-connected neural network layers. Such feedforward neural networks are universal function approximators of which the parameters and composed functions can be arranged conceptually into separate layers of neurons. In the input layer, each node is passed the value of one variable for each observation. The input layer is followed by a number of hidden layers with possibly varying numbers of neurons. Each observation represents a vector that passes through each of these layers and is transformed at each step until it reaches the output layer, where it has a possibly different shape, and its values represent the model prediction for the target variables. Each hidden layer has an associated weight matrix $\mathbf{W}$ and bias vector $\mathbf{b}$. What is meant by an observation $\mathbf{x}$ passing through such a layer is that a linear combination of the input vector or the output of the previous layer is taken, and an activation function $s(\cdot)$ is applied as in

$$z = s(\mathbf{x}\mathbf{W}^T + \mathbf{b}), \tag{3.3.1}$$

borrowing the notation from Vincent et al. (2008). With the activation function, we introduce nonlinearity into the model to give it the necessary flexibility. In this study, the ReLU activation function is used. It is defined as $s(x) = \max(0, x)$ and applied element-wise. In this formulation, each row in $\mathbf{W}$ and element in $\mathbf{b}$ is thus associated with a single neuron in the respective layer. This is a fully-connected neural network because, in each layer, every node is connected with every node of the preceding and following layers. This differentiates it from, e.g., a Convolutional Neural Network that s where some weights are shared between neurons.

To make the MP model generalize better, we make use of dropout layers between the linear layers (Srivastava et al., 2014). During training, these set a proportion of randomly selected activations to zero. This prevents excessive coadaptation of neurons and thus makes the model overfit less.

The outputs of the last layer - which has an optional activation function - are passed into a loss function which is minimized during training. For the DAE models (PDAE and ADAE), as well as to compute the final imputation loss, we use the mean-squared error loss defined as

$$\mathcal{L}(\mathbf{y}_i, \hat{\mathbf{y}}_i) = \left(||\mathbf{y}_i - \hat{\mathbf{y}}_i||_2\right)^2,$$

for the target observation $\mathbf{y}_i$ and the model prediction $\hat{\mathbf{y}}_i$. While the final imputation loss is naturally only computed on the variables that are missing for each observation, respectively, the loss function in training the ADAE model is computed on the full set of variables since a more focused loss calculation turned out to prevent the model from learning well.

Since the MP model predicts mask probabilities, we use a binary cross-entropy loss, as in

$$\mathcal{L}(\mathbf{y}_i, \hat{\mathbf{y}}_i) = -\mathbf{y}_i \log \hat{\mathbf{y}}_i - (1 - \mathbf{y}_i) \log(1 - \hat{\mathbf{y}}_i),$$

where all of the operations are applied element-wise to obtain a vector of loss values for target $\mathbf{y}_i$, which is then summed up to derive the scalar loss.

To train the model, the gradient with respect to the model parameters is backpropagated

through the network to update every weight a step in the direction that decreases the loss, with the size of the step being controlled by the learning rate hyperparameter. This gradient descent is stochastic because it is not executed after calculating the loss on every observation but rather on a random subsample of observations to achieve faster convergence. A more in-depth description of neural networks can be found in LeCun et al. (2015).

4 Numerical Results

In this section, we analyze the performance of the proposed model and examine the behavior of the different model components. For this, we first explain the different choices that were made for the experiments, followed by the experiments on the full *imputeLM* model. Then we show the performance of our implementation of the Denoising Autoencoder architecture. That section acts as a replication of Vincent et al. (2008), which is the reference paper for this thesis. The last two subsections take a closer look at the MP model and different aspects of the full model.

4.1 Experimental Setup

For the numerical experiments, we use three commonly available image datasets: CIFAR10 (Krizhevsky and Hinton, 2009), MNIST (LeCun, 1998) and FashionMNIST (Xiao et al., 2017) - example images can be found in Appendix A. CIFAR10 consists of 60,000 colour images which are 32x32 pixels. They contain images in ten classes (such as airplane, bird, cat and ship). For the experiments in this study, we convert the images to greyscale. MNIST contains 70,000 greyscale 28x28 pixels images of handwritten digits. FashionMNIST has the same size and format but contains images of articles sold on Zalando. For all three datasets, we convert the integer greyscale values to floats in [0, 1]. Unless specified otherwise, we split each dataset into fully-observed and partially-observed parts with a 60-40 split and run all experiments with a missingness level of 20%, i.e. 20% of elements in X missing. For the MAR, MNAR and QMNAR implementations, the specified levels are approximate since the correct missingness level has to be obtained via line search. Visual examples of the missingness mechanisms can be found in Section 4.4 and Appendix A. All results that are listed with a standard deviation come from a tenfold repetition of the experiments. To compute a validation loss during training for each model, we randomly split the dataset 80-20 into training and validation sets.

The performance of the *imputeLM* model is compared with a vanilla DAE model and a standard Variational Autoencoder (VAE, Kingma and Welling (2013)) as well as simple mean imputation. The vanilla DAE is trained with zero-masking noise for which the proportion of affected variables is set to the missingness level of the data. For further information on using VAE for imputation, we refer the reader to Pereira et al. (2020). Mean imputation is done by replacing the missing value with the mean value of all the observed instances of each variable, respectively. Other popular imputation methods such as kNN imputation (Pedregosa et al., 2011), MICE (Van Buuren and Oudshoorn, 1999) and softImpute (Mazumder et al., 2010) were also considered as benchmarks but were not able to finish the imputation within the time limit.

For the precise layer dimensions and hyperparameters of all models, we refer the reader to Appendix G. A description of the code can be found in Appendix H.

4.2 Analysis of imputeLM Model

In this section, we present the main experiment that showcases the performance of the *imputeLM* model compared with three benchmark imputation methods DAE, VAE and Mean imputation. Table 4.2.1 shows the RMSE on the imputation task of the respective models together with its standard deviation over the 10 experiment repetitions. We compare the performance for all six simulated missingness mechanisms at a missingness level of 20%. The dataset used in this experiment is the MNIST dataset. In Appendix D, results for the other two datasets and a 10% missingness level are reported.

Table 4.2.1: Imputation RMSE and standard deviation of imputeLM model and benchmark models on different missingness mechanisms simulated on MNIST dataset for a missingness level of 20%. The lowest RMSE are highlighted.

RMSE	Patch	Patches	MAR	MNAR	MCAR	QMNAR
imputeLM	**.269±.0030**	**.272±.0031**	**.121±.0147**	**.123±.0141**	**.110±.0011**	.153±.0038
DAE	.282±.0029	.295±.0027	.194±.0352	.211±.0389	.113±.0009	**.137±.0031**
VAE	.337±.0018	.349±.0023	.248±.0362	.263±.0394	.161±.0032	.170±.0047
Mean	.404±.0005	.437±.0005	.366±.0279	.373±.0279	.367±.0001	.345±.0026

This experiment shows how the imputeLM model significantly outperforms the other imputation methods in terms of RMSE across most types of missingness. For the PATCH, PATCHES, MAR and MNAR scenario, imputeLM shows the lowest RMSE over all three datasets and all missingness levels that were tested (see Section 4.5 and Appendix D for more). Only in the case of MCAR and QMNAR is there no clear winner between the imputeLM model and the benchmark DAE. This is in line with the theoretical justification of the imputeLM model. Since the idea of the imputeLM model is to be able to extract information from the missingness process, we would expect it to outperform for at least PATCH, PATCHES and MAR, as these do not depend on any unobserved information and provide substantial structure to be learned. For the MNAR case, the consistent difference in performance is not fully expected as the MNAR implementation here only directly depends on the missing variable itself, and one could think that it does not include a dependence on observed variables. However, by sampling the missingness proportions per variable from a multivariate normal distribution, which is estimated on the full empirical covariance matrix of the data, the included information about the relationship between all variables could provide a relevant level of structure that the imputeLM model can still learn. For QMNAR, one can argue that it is a pure version of MNAR, only depending on the one unobserved variable itself at a time. The difference in performance for these two scenarios can then be taken as a hint towards the hypothesis that in a general MNAR case, the imputeLM can still capture the effect of the *observed* variables on the missingness but struggles in the pure case. In other words, as long as the missingness does depend on any observed variables, the imputeLM will learn that structure and thus perform better, but if there is no such dependence, it has no advantage. For the MCAR scenario, it could be expected that the DAE performs better, possibly because its corruption mechanism is already highly similar to the missingness mechanism, and the additional moving parts in the imputeLM model (PDAE-Encoder and MP model) would have to be calibrated near-perfectly to match the effectiveness

of the simple corruption process.

A visualization of the imputation performance of all four models can be found in Figure 4.2.1. For illustration purposes, it shows imputation where the missingness mechanism is a fixed patch. It is thus not placed randomly on the vertical axis as in the PATCH mechanism but always in the same position. This is an easy pattern for the MP model to pick up, resulting in it generating the exact same patch in the training of the ADAE model. With the corruption process perfectly matching the missingness, the imputeLM model generates solutions to this inpainting task that are much more plausible than the other models. The setup of this experiment can be seen as the (unrealistic) scenario in which the difference between imputeLM and a vanilla DAE is maximized in terms of the divergence between the deterministic patch and the uniform corruption process. This scenario is included to show one end of the spectrum of scenarios in which the imputeLM model should have the largest advantage.

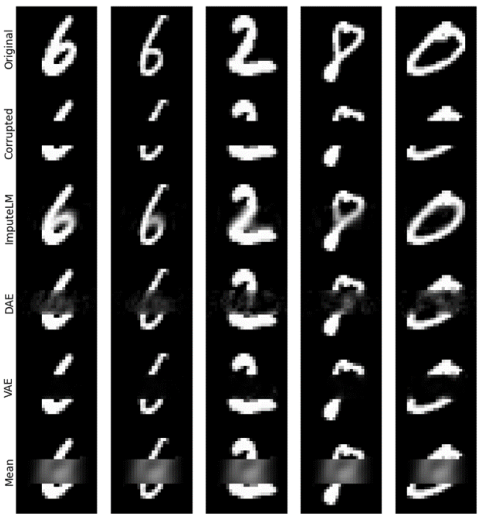

Figure 4.2.1: Examples of the imputation behavior of different models on several samples. The first row is the ground truth, the second row is with missingness applied, and the other rows show the respective imputed sample. The noise mechanism that was applied here is a fixed patch covering 30% of the image.

In Table 4.2.2, we evaluate the same models with the same missingness processes as before but now look at the effect on downstream task performance. For this, we use the classic MNIST classification task where each handwritten digit needs to be classified as the correct one from 0 to 9. To keep the additional complexity low, we only use a simple multinomial regression model to do this classification task on top of the imputed and fully observed images.

Table 4.2.2: Evaluation of models on downstream task: digit classification on imputed images. Displays classification accuracy and standard deviation. Simulated missingness level of 20% for different missingness mechanisms. The highest accuracies are highlighted. The dataset is MNIST.

Accuracy	Patch	Patches	MAR	MNAR	MCAR	QMNAR
imputeLM	.878±.0017	.880±.0014	**.915±.0014**	**.915±.0013**	**.916±.0006**	.907±.0011
DAE	.876±.0012	.878±.0016	.907±.0045	.905±.0053	**.916±.0004**	**.912±.0006**
VAE	.854±.0013	.852±.0013	.876±.0105	.871±.0112	.907±.0006	.892±.0020
Mean	**.883±.0009**	**.890±.0007**	.900±.0061	.897±.0066	.910±.0007	.903±.0013

What catches the eye about these results is that the MAR, MNAR, MCAR and QMNAR results align with the RMSE results, but the other two do not. For PATCH and PATCHES, it is the mean imputation that yields the best accuracy, while the same is never true for the RMSE. This also holds for the other datasets and missingness levels. One possible explanation could be that the more sophisticated imputation models make a stronger guess about the underlying digit in the imputation. This could lead to situations where if the patch covers the complete middle part of the digit and multiple inpainting solutions seem plausible to the human eye, the imputeLM model would make an imputation that matches only one of those closely. The downstream classification model would then confidently follow the direction indicated by the imputeLM model and possibly misclassify. The more blurry, unspecific imputation from the mean imputation does not take a stance as to the concrete underlying data instance whatsoever so it also is unlikely to misguide the classification model.

4.3 Denoising Autoencoders (Replication)

In this section, we briefly showcase the reconstruction performance of our own Denoising Autoencoder implementation. This serves as an approximate replication of the paper by Vincent et al. (2008), which introduces the Denoising Autoencoder idea and serves as the reference paper for this thesis. In the left part of Figure 4.3.1, we show example images of the MNIST dataset together with their corrupted version that we generate for the training of the DAE. The last row represents the output of the DAE, which is a reconstruction of the first row, only being given the input of the second row. One can see that the model learns to capture the characteristic shape of the digits and also replace the masked values accurately.

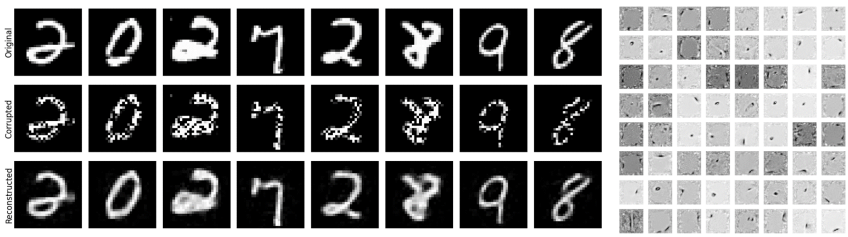

Figure 4.3.1: Visualization of Denoising Autoencoder reconstructions (left) and first layer weights per neuron. The dataset is MNIST.

As we mentioned in the theory section of the DAE, its main purpose is usually to extract meaningful higher-level representations from the input data. Vincent et al. (2008) demonstrate that their Denoising Autoencoder does that by visualizing a few sets of weights of the hidden layers. In their study, they find that black dots and edge-like structures in the weights indicate the model learning meaningful features such as edge- and stroke-detectors. Our implementation of the DAE shows similar behavior in the first layer weights displayed in the second subfigure of Figure 4.3.1.

4.4 Missingness Prediction Task

In this section, we examine the MP model itself. In Table 4.4.1, different performance measures with respect to the generated masks are shown. We show here the results for the FashionMNIST

dataset (see Appendix D for more). For the fully-observed samples of the dataset, we generate masks with different simulated missingness mechanisms and compare those masks to the ones generated by an MP model with a PDAE-Encoder in front (MP) and without a PDAE-Encoder in front (MPnoEnc) trained on the respective partially-observed samples. The comparison is in terms of RMSE between the generated and true masks, correlation between the masks and exact match ratio when regarding the mask generation as a multi-label classification task.

The model with PDAE-Encoder outperforms the one without in all measures and all mechanisms except for the RMSE for MNAR and MCAR. Especially for the PATCH scenario the PDAE-Encoder results in better generated masks. These results can be taken as an indication that such an Encoder can help substantially but also does not hurt otherwise as it does not perform worse in all measures for any of the mechanisms. For the correlation, one would not expect there to be a significant correlation in the MCAR case as the MP model is supposed to learn to emulate the random uniform distribution, which, if done correctly, would result in the generated mask being another independent and identically distributed draw from the same distribution thus implying a correlation of zero.

Table 4.4.1: Evaluation of masks generated by the MP model with and without a PDAE-Encoder in front. All three performance measures are computed between the MP-generated mask and the mask generated by the simulated missingness mechanism for the same image. Missingness level is 20%, dataset is FashionMNIST.

Criterion	Model	Patch	MNAR	MCAR
RMSE	MP Enc	0.176	0.359	0.100
	MP	0.265	0.353	0.065
Correlation	MP Enc	0.218	0.061	-0.002
	MP	-0.678	0.056	-0.032
Accuracy	MP Enc	0.625	0.644	0.620
	MP	0.560	0.640	0.606

The fact that the correlation is negative for PATCH without a PDAE-Encoder can also be seen visually in Figure 4.4.1. In this figure, we visualize an example of a simulated missingness pattern together with the mask generated by the two models. The last three columns show the same but averaged over all masks, corresponding to an observation in the fully-observed samples of the FashionMNIST dataset. If we look at the examples of the PATCH scenario, we see that both models generate a noisy version of what resembles a patch. In the average case, the simulated patch is found more often vertically in the middle, but the MP models seem to place their patch preferably at the top or bottom. Especially for the MP model without PDAE-Encoder, we see that the average image is almost a reversal of the simulated average. This is a visual confirmation of the negative correlation seen in the previous table. For the MP model with PDAE-Encoder, more patches seem to be placed in the middle, in line with the non-negative correlation.

One can imagine an extensive placement of the patches at the top or bottom to be problematic for the imputation and downstream task performance, especially for MNIST and FashionMNIST. This is because these images are only black in those areas; therefore, covering these areas as a corruption process does not pose a meaningful reconstruction task in the training of the

ADAE model. For the FashionMNIST dataset, this argument is supported by the difference in performance attributed to the PDAE-Encoder model in Table 4.5.1 keeping in mind that making use of the PDAE-Encoder seems to generate more patches in the middle.

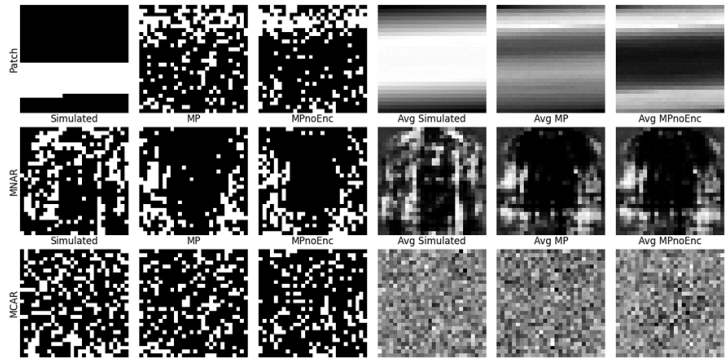

Figure 4.4.1: Simulated and learned masks for different missingness mechanisms. White pixels are missing, and black are observed. The first column shows one true mask, the second column the generation of the MP model, third column the generation of an MP model without PDAE-Encoder in front. The last three columns show a mask that represents the average over all generated masks for these three models. The dataset is FashionMNIST.

For the MNAR and MCAR scenarios, we do not have a substantial negative correlation, and this is also represented in the visualization. One can see that both MP and MPnoEnc generate masks that adhere to a visually non-random structure in the MNAR case, albeit the exact structure and missingness level do not perfectly match the simulated mask. In the MCAR case, the model seems to replicate the completely random nature of the mask well and even generates a similar missingness level.

When looking at the same visualization but for the other datasets in Appendix D, the effect of the PDAE-Encoder seems less clear. While it never hurts the accuracy of the generated masks, it does yield a small negative correlation for CIFAR10, where the model without PDAE-Encoder gives a positive one. Thus, we cannot say that the theoretical advantage of using the PDAE-Encoder materializes in practice in general.

However, taking a broader view of these MP model results, one can see that although the exact pattern generation is imperfect in general, the MP model-generated averages all look substantially different across the different missingness mechanisms and reasonably close in structure to the simulated average, admitting the interpretation that the model takes a step in the direction of learning the missingness. This holds for all three datasets equivalently.

4.5 Further Experiments

To analyze the behavior of the imputeLM model further, we provide four more experiments in this section. Table 4.5.1 presents the results of the first two. The first three rows compare the imputation performance of the imputeLM model with a PDAE-Encoder in front, without a PDAE-Encoder and with a PDAE-Encoder but zero as the replacement value instead of a uniform random one.

We can see that the model with PDAE-Encoder and uniform random replacement performs best across all three mechanisms. This is expected due to the analysis of the MP model with

and without the Encoder. In Section 3.2.3, we argued how a uniform replacement should be better than a fixed value, which is supported by these data.

Table 4.5.1: Imputation RMSE results of the following experiments: The first row is a regular imputeLM model with uniform random replacement for the MP model. The second row is the same model but without PDAE-Encoder. The third row is a complete imputeLM model but with zero replacement for the MP model. The last two rows compare how much a vanilla DAE can improve by using the true underlying missingness mechanism as a corruption process. Missingness level is 20%, dataset is FashionMNIST.

RMSE	Patch	MNAR	MCAR
imputeLM	**.267±.006**	**.128±.019**	**.116±.001**
imputeLM no Enc	.297±.007	.144±.023	.132±.001
imputeLM zero	.314±.007	.147±.034	**.116±.001**
DAE	.382±.008	.252±.088	**.110±.002**
DAE true	**.149±.001**	**.237±.056**	.112±.002

The second experiment in this table (last two rows) evaluates one of the core hypotheses of this imputation approach: The hypothesis that the imputation model can perform better if trained on the right missingness mechanism as its corruption process. We compare the vanilla DAE with a DAE that is trained on the true simulated missingness mechanism. While this mechanism is unknown in practice, we can use it here to establish an upper bound on how well we can do if the MP model were to generate perfectly similar masks. We can see that for PATCH and MNAR, the knowledge of the true mechanism results in a significant increase in imputation performance, while this is not the case for MCAR.

Finally, we provide a sensitivity analysis of our results with respect to the missingness level and the size of the partially-observed dataset in relation to the fully-observed set in Figure 4.5.1. We can see that the imputeLM model outperforms the other approaches across various parameter choices for the FashionMNIST MNAR scenario.

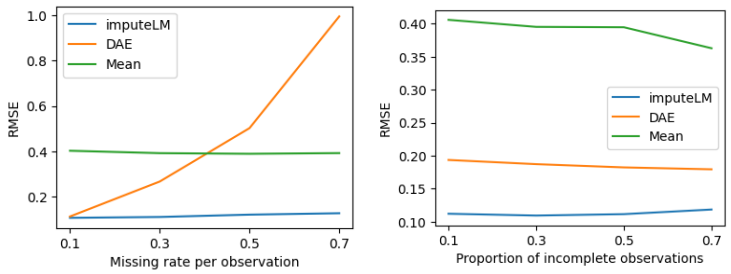

Figure 4.5.1: Imputation RMSE for varying missing rates and proportion of incomplete observations. In (a) the proportion of incomplete observations is 0.4 and in (b) the missing rate is 0.2. VAE was excluded due to inferior performance in the main results. The dataset is FashionMNIST and the missingness mechanism is MNAR.

5 Conclusion

In this paper, we propose a new imputation model that modifies the well-known Denoising Autoencoder model. The question of how to deal with missing values is encountered frequently by anyone dealing with real-world datasets. Especially for high dimensional datasets it is not a possibility to discard all incomplete samples, thus the missing values need to be imputed

with reasonable values. Common challenges for previous imputation methods are that many of them estimate one supervised learning model per variable, which makes them unsuitable for high-dimensional cases such as image datasets. This disqualifies them from a large array of image-related reconstruction tasks that can also be tackled in the imputation framework. Many deep learning-based imputation methods - such as using a standard DAE or Variational Autoencoder for imputation - have the problem that they either suffer from an unrealistic data scenario, relying on data not available in practice or they train only on the fully-observed portion of observations (Yoon et al., 2018). This leaves out valuable information that one can gain from the missingness itself.

The approach taken in this study solves these problems. It applies just as well to high-dimensional datasets as to regular tabular datasets and is based on a realistic dataset scenario. We propose the *imputeLM* method, which adapts a Denoising Autoencoder model, training it on a learned model of the missingness mechanism. In this way, it takes into account all available information and provides for a more flexible imputation model.

We demonstrate the performance of the new model on three image datasets for several missingness configurations and present an analysis as to which part of the model acts in which way. The model performs better than the benchmark models consistently for all levels of missingness and dataset configurations for all but the MCAR and QMNAR scenarios. We show that the MP component of the imputeLM model can learn different missingness scenarios and that using an additional Encoder in front of the MP model can help it generate better masks.

The approach presented here has several limitations. Training three separate models and chaining them together makes the imputation task a more complicated endeavor that is less stable than simpler imputation models. For optimal performance, one would need to tune each hyperparameter of the three models, which would require a considerable investment of resources. However, the fact that in this study - where the models were not tuned extensively - performance increases could already be observed indicates that for situations in which a proper imputation is crucial, and the missingness shows a substantial amount of structure, using the more thorough approach presented here could pay off. Another limitation is that the experiments of this study are limited to datasets of small images.

For further research, one could easily extend this approach to other datasets and datatypes. Since the interaction between learning missingness patterns and using them for imputation makes for a conceptual framework that can be filled with a neural network of choice, one could explore different architectural choices, such as Convolutional Neural Networks or Residual Networks for different applications. An interesting direction could also be to use a Generative Adversarial Network (GAN) as a missingness predictor model. While a proper discussion of this variant is out of scope for this study, first results showed promising performance increases over the regular model - at the cost of GANs being more unstable and difficult to train. Another possibility is to use this framework for multiple imputation. One can also extend this research by using the model to perform Sequential Active Information Acquisition (SAIA) as in Ma et al. (2018). Finally, it could be of interest to explore the connection between learning missingness and the idea of learning generated noise and removing it as in diffusion models.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., and Isard, M. (2016). Tensorflow: a system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283.

Bengio, Y., Yao, L., Alain, G., and Vincent, P. (2013). Generalized denoising auto-encoders as generative models. *Advances in neural information processing systems*, 26.

Biewald, L. (2020). Experiment tracking with weights and biases. Software available from wandb.com.

Boquet, G., Morell, A., Serrano, J., and Vicario, J. L. (2020). A variational autoencoder solution for road traffic forecasting systems: Missing data imputation, dimension reduction, model selection and anomaly detection. *Transportation Research Part C: Emerging Technologies*, 115:102622.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585:357–362.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95.

Ipsen, N. B., Mattei, P.-A., and Frellsen, J. (2020). not-miwae: Deep generative modelling with missing not at random data. *arXiv preprint arXiv:2006.12871*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images.

LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Little, R. J. and Rubin, D. B. (2019). *Statistical analysis with missing data*, volume 793. John Wiley & Sons.

Ma, C., Tschiatschek, S., Palla, K., Hernández-Lobato, J. M., Nowozin, S., and Zhang, C. (2018). Eddi: Efficient dynamic discovery of high-value information with partial vae. *arXiv preprint arXiv:1809.11142*.

Ma, Q., Lee, W.-C., Fu, T.-Y., Gu, Y., and Yu, G. (2020). Midia: exploring denoising autoencoders for missing data imputation. *Data Mining and Knowledge Discovery*, 34:1859–1897.

Mattei, P.-A. and Frellsen, J. (2019). Miwae: Deep generative modelling and imputation of incomplete data sets. In *International conference on machine learning*, pages 4413–4423. PMLR.

Mazumder, R., Hastie, T., and Tibshirani, R. (2010). Spectral regularization algorithms for learning large incomplete matrices. *The Journal of Machine Learning Research*, 11:2287–2322.

McCoy, J. T., Kroon, S., and Auret, L. (2018). Variational autoencoders for missing data imputation with application to a simulated milling circuit. *IFAC-PapersOnLine*, 51(21):141–146.

McKinney, W. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX.

Muzellec, B. (2019). Data imputation using optimal transport. https://github.com/BorisMuzellec/MissingDataOT.

Muzellec, B., Josse, J., Boyer, C., and Cuturi, M. (2020). Missing data imputation using optimal transport. In *International Conference on Machine Learning*, pages 7130–7140. PMLR.

NVIDIA, Vingelmann, P., and Fitzek, F. H. (2020). Cuda, release: 10.2.89.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., and Dubourg, V. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Pereira, R. C., Santos, M. S., Rodrigues, P. P., and Abreu, P. H. (2020). Reviewing autoencoders for missing data imputation: Technical trends, applications and outcomes. *Journal of Artificial Intelligence Research*, 69:1255–1285.

Raschka, S. (2019). Deeplearning models. https://github.com/rasbt/deeplearning-models.

Rubin, D. B. (1976). Inference and missing data. *Biometrika*, 63(3):581–592.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Stekhoven, D. J. and Bühlmann, P. (2012). Missforest—non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118.

TorchVision maintainers, c. (2016). Torchvision: Pytorch's computer vision library. https://github.com/pytorch/vision.

Van Buuren, S. and Oudshoorn, K. (1999). *Flexible multivariate imputation by MICE*. Leiden: TNO.

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Yoon, J., Jordon, J., and Schaar, M. (2018). Gain: Missing data imputation using generative adversarial nets. In *International conference on machine learning*, pages 5689–5698. PMLR.

A Data

For reference, we include a few example images from the MNIST, FashionMNIST and CIFAR10 datasets in Figure A.1.

Figure A.1: MNIST, FashionMNIST and CIFAR10 examples.

Figure A.2 shows some examples of the PATCHES and QMNAR missingness mechanisms applied to CIFAR10. PATCH, MNAR and MCAR are also shown in Figure 4.4.1. Visually the MAR missingness is comparable to MNAR.

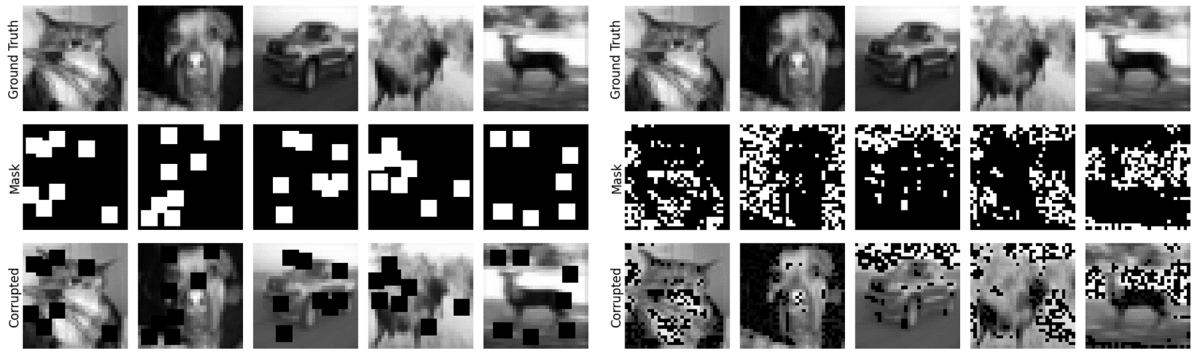

Figure A.2: Missingness examples.

B Derivation Missingness Simulation

In this section, we show the equivalence between our formulation of the MAR and MNAR processes and those taken from Yoon et al. (2018). We denote the probability that observation x_{ij} is missing with the shorthand p_{ij}. For the MAR process, this is defined as

$$p_{ij} = \frac{p_j \cdot n \cdot \exp(-\Psi(i,j))}{\sum_{l=1}^{n} \exp(-\Psi(l,j))}, \tag{B.0.1}$$

where

$$\Psi(i,j) = \sum_{k<j} \left(w_k s_{ik} x_{ik} + b_k(1 - s_{ik})\right). \tag{B.0.2}$$

We first rewrite $\Psi(i,j)$ in matrix notation. For this, we define the vector Ψ_j which has as elements $\Psi(i,j), \forall i \in \mathcal{I}$, i.e. the scores corresponding to the variable j across all observations.

We can write it as

$$\Psi_j = \operatorname{diag}\left(\begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,j-1} \\ x_{2,1} & x_{2,2} & \dots & x_{2,j-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \dots & x_{n,j-1} \end{bmatrix} \begin{bmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_{j-1} \end{bmatrix} \begin{bmatrix} s_{1,1} & s_{1,2} & \dots & s_{1,j-1} \\ s_{2,1} & s_{2,2} & \dots & s_{2,j-1} \\ \vdots & \vdots & \ddots & \vdots \\ s_{n,1} & s_{n,2} & \dots & s_{n,j-1} \end{bmatrix}^T\right)$$
$$+ \begin{bmatrix} b_1 & b_2 & \dots & b_{j-1} \end{bmatrix} \begin{bmatrix} 1-s_{1,1} & 1-s_{1,2} & \dots & 1-s_{1,j-1} \\ 1-s_{2,1} & 1-s_{2,2} & \dots & 1-s_{2,j-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1-s_{n,1} & 1-s_{n,2} & \dots & 1-s_{n,j-1} \end{bmatrix}^T .$$

We denote the matrices as $X, W, S, \mathbf{b}$ and S^* in order of appearance in the previous equation. We can then write the equation in terms of submatrices *new* and *old*. Old refers to all columns $1, \dots, j-2$, and new denotes the column $j-1$. The submatrices can then be written as a separate summation of *old* and *new* parts as in

$$\begin{aligned}
\Psi_j = &\ \operatorname{diag}\left(\begin{bmatrix} X_{old} & X_{new} \end{bmatrix} \begin{bmatrix} W_{old} & O \\ O & W_{new} \end{bmatrix} \begin{bmatrix} S_{old} & S_{new} \end{bmatrix}^T\right) + \begin{bmatrix} \mathbf{b}_{old} & \mathbf{b}_{new} \end{bmatrix} \begin{bmatrix} S^*_{old} & S^*_{new} \end{bmatrix}^T \\
= &\ \operatorname{diag}\left(\begin{bmatrix} X_{old}W_{old} & X_{new}W_{new} \end{bmatrix} \begin{bmatrix} S^T_{old} \\ S^T_{new} \end{bmatrix}\right) + \mathbf{b}_{old}S^{*T}_{old} + \mathbf{b}_{new}S^{*T}_{new} \\
= &\ \operatorname{diag}\left(X_{old}W_{old}S^T_{old} + X_{new}W_{new}S^T_{new}\right) + \mathbf{b}_{old}S^{*T}_{old} + \mathbf{b}_{new}S^{*T}_{new} \\
= &\ \operatorname{diag}\left(X_{old}W_{old}S^T_{old}\right) + \mathbf{b}_{old}S^{*T}_{old} + \operatorname{diag}\left(X_{new}W_{new}S^T_{new}\right) + \mathbf{b}_{new}S^{*T}_{new}.
\end{aligned}$$

Then defining $A = XW$ with the same column subsets as before, we obtain

$$\Psi_j = \operatorname{diag}(A_{1:j-2}S^T_{1:j-2}) + \mathbf{b}_{1:j-2}S^{*T}_{1:j-2} + \operatorname{diag}(A_{j-1}S^T_{j-1}) + \mathbf{b}_{j-1}S^{*T}_{j-1}), \tag{B.0.3}$$

which is the formulation introduced above and used for the implementation.

One difference to note is that Yoon et al. (2018) define p_j as the average *missing* rate of the j-th variable but p_{ij} as the probability that x_{ij} is *not* missing. To make it more consistent, we interpret p_{ij} as the probability that x_{ij} *is* missing. This should not make much of a difference, especially since the exact parameters of the sigmoid, which determine the missingness level, are chosen via line search. Regarding the efficiency of this computation, it is worth noting that with the matrices inside the $\operatorname{diag}(\cdot)$ operation, we still calculate several unnecessary values that are simply discarded by extracting only the diagonal. However, in practice, this formulation is still

quicker than a for-loop implementation.

For the MNAR process, the computation boils down to finding A and using one column after another.

C Model Visualization

In this section, we provide a full illustration of the sequential training process of the imputeLM model. In the illustration, a dataset is shown in which the first two observations are fully observed, and the last two observations have one missing value each. For each part of the imputeLM model, we show the training setup followed by the actual imputation in Figures C.1-4. Some of the design choices are inspired by the model visualization in Yoon et al. (2018).

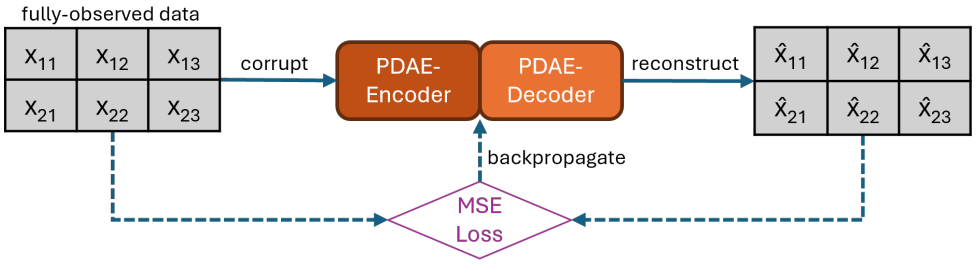

Figure C.1: Training setup of the PDAE-Encoder model.

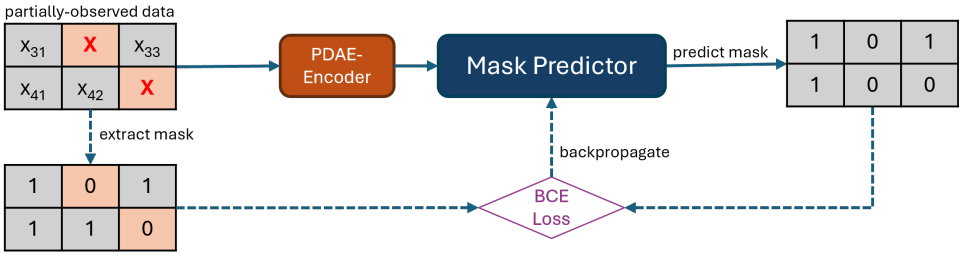

Figure C.2: Training setup of the Missingness Predictor (MP) model.

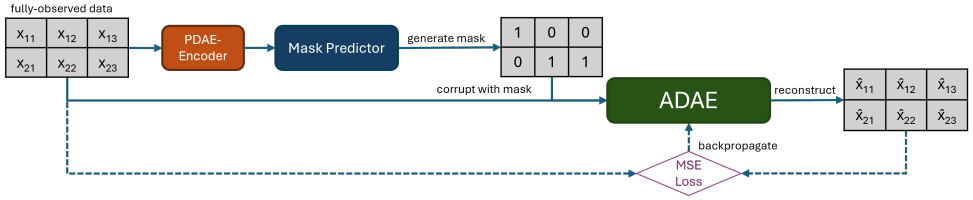

Figure C.3: Training setup of the Adapted Denoising Autoencoder (ADAE), which is used to impute the missing values.

Figure C.4: Demonstration of imputation task.

D Additional Results

Table D.1: RMSE for all datasets. Missingness levels are listed as percentages in the header.

	RMSE	Patch (10%)	Patch (20%)	Patches (10%)	Patches (20%)	MAR (10%)	MAR (20%)	MNAR(10%)	MNAR (20%)	MCAR (10%)	MCAR (20%)	QMNAR(10%)	QMNAR(20%)
MNIST	imputeLM	**.231±.0016**	**.269±.0030**	**.268±.0023**	**.272±.0031**	**.127±.0153**	**.121±.0147**	**.129±.0147**	**.123±.0141**	**.107±.0014**	**.110±.0011**	**.144±.0083**	.153±.0038
	DAE	.270±.0019	.282±.0029	.315±.0025	.295±.0027	.242±.0406	.194±.0352	.257±.0438	.211±.0389	.130±.0019	.113±.0009	.145±.0071	**.137±.0031**
	VAE	.299±.0028	.337±.0018	.343±.0030	.349±.0023	.262±.0437	.248±.0362	.278±.0479	.263±.0394	.133±.0032	.161±.0032	**.144±.0106**	.170±.0047
	Mean	.386±.0006	.404±.0005	.439±.0007	.437±.0005	.405±.0339	.366±.0279	.413±.0328	.373±.0279	.367±.0003	.367±.0001	.353±.0133	.345±.0026
FMNIST	imputeLM	**.192±.0038**	**.227±.0031**	**.191±.0027**	**.199±.0055**	**.122±.0139**	**.127±.0167**	**.122±.0154**	**.128±.0194**	**.111±.0011**	.116±.0010	.162±.0069	.220±.0063
	DAE	.268±.0051	.308±.0059	.236±.0061	.223±.0058	.238±.0759	.228±.0783	.257±.0843	.252±.0884	.115±.0041	**.110±.0016**	**.151±.0063**	**.188±.0056**
	VAE	.267±.0153	.360±.0108	.251±.0164	.286±.0142	.247±.0732	.268±.0741	.264±.0818	.286±.0810	.135±.0042	.169±.0048	.169±.0074	.226±.0038
	Mean	.426±.0004	.429±.0005	.452±.0005	.452±.0004	.415±.0291	.406±.0264	.418±.0291	.407±.0264	.416±.0002	.416±.0001	.417±.0085	.425±.0043
CIFAR10	imputeLM	**.197±.0027**	**.219±.0043**	**.183±.0009**	**.186±.0021**	**.131±.0109**	**.138±.0094**	**.134±.0155**	**.149±.0203**	.112±.0009	.118±.0019	.223±.0073	.314±.0063
	DAE	.263±.0133	.336±.0084	.209±.0098	.200±.0047	.248±.0539	.261±.0482	.267±.0622	.289±.0571	**.101±.0036**	**.107±.0018**	**.196±.0067**	**.268±.0054**
	VAE	.293±.0064	.380±.0051	.240±.0069	.266±.0065	.245±.0448	.283±.0443	.258±.0505	.300±.0485	.124±.0090	.160±.0103	.216±.0161	.313±.0068
	Mean	.333±.0006	.331±.0006	.330±.0006	.330±.0002	.335±.0058	.335±.0049	.335±.0059	.335±.0050	.336±.0001	.336±.0001	.418±.0010	.419±.0006

Table D.2: Classification accuracy for all datasets. Missingness levels are listed as percentages in the header.

	Accuracy	Patch (10%)	Patch (20%)	Patches (10%)	Patches (20%)	MAR (10%)	MAR (20%)	MNAR(10%)	MNAR (20%)	MCAR (10%)	MCAR (20%)	QMNAR(10%)	QMNAR(20%)
MNIST	imputeLM	.900±.0014	.878±.0017	.896±.0013	.880±.0014	**.916±.0011**	**.915±.0014**	**.916±.0010**	**.915±.0013**	**.916±.0005**	**.916±.0006**	.911±.0010	.907±.0011
	DAE	.897±.0007	.876±.0012	.892±.0010	.878±.0016	.907±.0039	.907±.0045	.906±.0045	.905±.0053	**.916±.0009**	**.916±.0004**	**.913±.0007**	**.912±.0006**
	VAE	.885±.0012	.854±.0013	.878±.0015	.852±.0013	.885±.0086	.876±.0105	.882±.0098	.871±.0112	.912±.0006	.907±.0006	.907±.0014	.892±.0020
	Mean	**.905±.0006**	**.883±.0009**	**.903±.0010**	**.890±.0007**	.905±.0043	.900±.0061	.903±.0049	.897±.0066	.914±.0007	.910±.0007	.911±.0009	.903±.0013
FMNIST	imputeLM	.796±.0022	.784±.0021	.796±.0015	.789±.0020	**.835±.0031**	**.832±.0054**	**.835±.0040**	**.831±.0072**	.838±.0019	.836±.0015	.834±.0012	.825±.0018
	DAE	.822±.0015	.808±.0015	.820±.0015	.812±.0019	.831±.0079	.828±.0098	.830±.0102	.825±.0126	**.840±.0018**	**.839±.0022**	**.838±.0026**	**.832±.0021**
	VAE	.807±.0014	.788±.0017	.804±.0021	.792±.0019	.804±.0105	.795±.0140	.801±.0128	.791±.0191	.833±.0015	.828±.0013	.831±.0016	.804±.0025
	Mean	**.835±.0011**	**.824±.0015**	**.832±.0010**	**.824±.0014**	.831±.0060	.826±.0079	.830±.0072	.825±.0093	.838±.0020	.835±.0019	.836±.0015	.829±.0015
CIFAR10	imputeLM	.244±.0061	.240±.0066	.251±.0063	.244±.0064	.279±.0053	.277±.0049	.278±.0054	.274±.0053	.282±.0089	.280±.0085	.282±.0058	.264±.0064
	DAE	.258±.0062	.241±.0070	.265±.0054	.257±.0052	.281±.0077	.278±.0077	.280±.0076	.275±.0076	.285±.0075	.284±.0074	.285±.0067	.269±.0070
	VAE	.259±.0067	.246±.0064	.266±.0058	.257±.0063	.273±.0076	.266±.0094	.271±.0085	.264±.0101	.285±.0061	.284±.0057	.284±.0074	.260±.0053
	Mean	**.284±.0053**	**.278±.0054**	**.284±.0059**	**.281±.0052**	**.283±.0073**	**.281±.0082**	**.283±.0075**	**.281±.0076**	**.287±.0055**	**.285±.0054**	**.288±.0053**	**.281±.0077**

Table D.3: Evaluation of masks generated by the MP model with and without a PDAE-Encoder in front. All three performance measures are computed between the MP-generated mask and the mask generated by the simulated missingness mechanism for the same image. The missingness level is 20%.

Criterion	Model		Patch	MNAR	MCAR		Patch	MNAR	MCAR		Patch	MNAR	MCAR
RMSE	MP Enc	MNIST	0.237	0.308	0.100	FMNIST	0.176	0.359	0.100	CIFAR10	0.178	0.362	0.101
	MP		0.291	0.306	0.091		0.265	0.353	0.065		0.143	0.366	0.077
Correlation	MP Enc		-0.416	0.092	0.005		0.218	0.061	-0.002		-0.114	0.114	-0.003
	MP		-0.792	0.092	-0.031		-0.678	0.056	-0.032		0.210	0.109	0.037
Accuracy	MP Enc		0.591	0.652	0.620		0.625	0.644	0.620		0.613	0.656	0.620
	MP		0.546	0.649	0.616		0.560	0.640	0.606		0.588	0.657	0.611

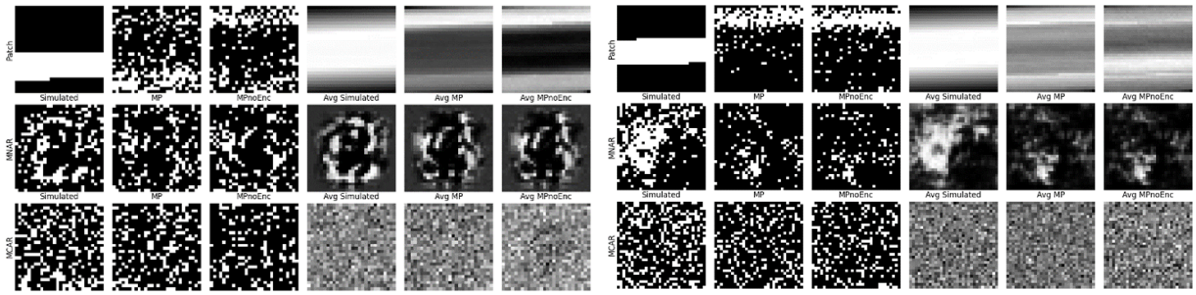

Figure D.1: Simulated and learned masks for different missingness mechanisms. White pixels are missing, and black are observed. The first column shows one true mask, the second column the generation of the MP model, third column the generation of an MP model without a PDAE-Encoder in front. The last three columns show a mask that represents the average over all generated masks for these three models. The datasets are MNIST (left) and CIFAR10.

E imputeLM Pseudo Code

This section gives a detailed description of how to train and use the imputeLM model. Algorithm 1 contains the pseudo-code the notation of which is inspired by Yoon et al. (2018).

For the model setup and training, it is worth noting that in this model chain, one has to pay close attention to which observations are used at which point and in which form (encoded or with replacement values) and with matching replacement values for training and inference. For example, the ADAE model is trained with the mask applied as zero-masking noise (see above), so the partially observed samples used for inference with the ADAE model should have the missing values replaced with zero, while the same observations used for the training of the MP model should have the missing values replaced in the same way as the PDAE-Encoder was trained on (which in this study is uniform replacement).

Algorithm 1 imputeLM Training and Imputation

Choose batch sizes k_E, k_{MP}, k_{ADAE}
Choose number of epochs r_E, r_{MP}, r_{ADAE}
(1) Train PDAE-Encoder $l_\eta(\mathbf{x})$
Choose corruption proportion p
for iterations $\leq r_E$ **do**
  while not all observations in $\mathcal{I}_o$ have been seen **do**
    Draw k_E samples from dataset $\{\mathbf{x}_i\}_{i \in \mathcal{I}_o}$
    for $i = 1, \ldots, k_E$ **do**
      Corrupt sample by setting a proportion p of variables to iid draws from $\mathcal{U}(0,1)$ to obtain $\mathbf{x}_i^*$
      $\hat{\mathbf{x}}_i \leftarrow m_\phi(l_\eta(\mathbf{x}_i^*))$
    end for
    Update ϕ, η using Stochastic Gradient Descent on $\nabla_{\phi,\eta} - \sum_{i=1}^{k_E} \mathcal{L}(\mathbf{x}_i, \hat{\mathbf{x}}_i)$
  end while
end for
(2) Train MP model $h_\lambda(\mathbf{z})$
for iterations $\leq r_{MP}$ **do**
  while not all observations in $\mathcal{I}_m$ have been seen **do**
    Draw k_{MP} samples from dataset $\{\tilde{\mathbf{x}}_i\}_{i \in \mathcal{I}_m}$
    for $i = 1, \ldots, k_{MP}$ **do**
      Encode sample: $\mathbf{z}_i \leftarrow l_\eta(\tilde{\mathbf{x}}_i)$
      Predict mask: $\hat{\mathbf{s}}_i \leftarrow h_\lambda(\mathbf{z}_i)$
    end for
    Update λ using Stochastic Gradient Descent on $\nabla_\lambda - \sum_{i=1}^{k_{MP}} \mathcal{L}(\mathbf{s}_i, \hat{\mathbf{s}}_i)$
  end while
end for
(3) Train ADAE model $g_\omega(f_\delta(\mathbf{x}))$
for iterations $\leq r_{ADAE}$ **do**
  while not all observations in $\mathcal{I}_o$ have been seen **do**
    Draw k_{ADAE} samples from dataset $\{\mathbf{x}_i\}_{i \in \mathcal{I}_o}$
    for $i = 1, \ldots, k_{ADAE}$ **do**
      Generate mask from encoded sample: $\hat{\mathbf{s}}_i \leftarrow h_\lambda(l_\eta(\mathbf{x}_i))$
      Use mask to corrupt sample: $\tilde{\mathbf{x}}_i \leftarrow \hat{\mathbf{s}}_i \odot \mathbf{x}_i$
      Reconstruct sample: $\hat{\mathbf{x}}_i \leftarrow g_\omega(f_\delta(\tilde{\mathbf{x}}_i))$
    end for
    Update ω, δ using Stochastic Gradient Descent on $\nabla_{\omega,\delta} - \sum_{i=1}^{k_{ADAE}} \mathcal{L}(\mathbf{x}_i, \hat{\mathbf{x}}_i)$
  end while
end for
(4) Impute missing values
for all $i \in \mathcal{I}_m$ **do**
  Reconstruct $\hat{\mathbf{x}}_i \leftarrow g_\omega(f_\delta(\tilde{\mathbf{x}}_i))$
  Replace x_{ij} by $\hat{x}_{ij}$, for all i, j s.t. $s_{ij} = 0$
end for

F GAN as MP Model

Replacing the generic MP model with a Generative Adversarial Network (GAN, (Goodfellow et al., 2014)) model to produce realistic missingness patterns could increase the overall performance of the imputeLM model additionally. Preliminary experiments showed promising results in visual inspection of the masks that a GAN can produce. However, these experiments are not followed up on or included in this study due to time and space constraints paired with the complexity of tuning a GAN. In this section, we briefly describe one possible approach to training a GAN in this context.

In general, a GAN consists of two models that are trained adversarially. A generator model (G) generates samples that are as realistic as possible in the context of a dataset of true samples. A discriminator model (D) is a classification model with the aim of distinguishing the generated samples of G from the set of real samples. The idea is that G can trick D the best when the generated samples are the most realistic, and D can classify the most accurately if it recognizes even minor differences. In this interplay, G can learn to generate highly realistic samples.

In our case, G needs to generate (binary) missingness masks and the discriminator tries to distinguish them from the actual missingness masks found in the partially-observed dataset. To mimic an MCAR process, it would then be theoretically sufficient to provide G pure noise as the input and let it generate arbitrary masks. Similarly, the discriminator only receives the pure mask as input. However, then the generator could not learn to extract the information that is given in MAR and MNAR processes as it does not see the observed variables and thus cannot learn to infer missingness from their interplay. We should, therefore, condition both the generator and the discriminator on the actual observation.

A simple way to do this would be to concatenate the observation vector to the noise vector in the input of the generator. The generated mask could also be concatenated to the full observation for the input of the discriminator. However, at this point, a similar problem as for the regular MP model appears, which is that of recognizing replacement values. Because we also need to pass the partially-observed samples as conditioning information to the discriminator (together with the corresponding mask), the discriminator will likely not learn to distinguish real masks from generated masks based on the realism of the mask, but rather it would learn to recognize whether an observation is fully observed or has missing values (replaced with some value or noise). We thus need to encode the conditioning value in a way that ideally does not give the discriminator the information about the missingness. We propose two ways to do this.

A first possibility is not to concatenate the conditioning information to the mask and instead use the mask to corrupt the actual observation. The discriminator would then only see corrupted observations - be they corrupted by reality or by the generator. This could provide a more accurate learning signal for G since D would likely be able to spot unrealistic masks irrespective of the underlying information. However, in practice, this did not converge to realistic masks. What could be observed is that the generator would initially learn accurate masks but eventually begin to cover the whole image. One possible interpretation of this could be that a sufficiently strong discriminator seems to learn to distinguish which observations are fully observed vs. partially observed based on the (corrupted) conditioning information. Instead of evaluating the interplay of mask and observation, it would remember which observations are real. In response

to that, G learns to corrupt the observation so much that D cannot recognize it anymore. This results in generated masks that remove almost all information from the observation instead of being similar to the observed missingness. It is possible that with a more granular tuning of the parameters and relative strength of both models, a balance could be achieved where this collapse does not happen.

The second possibility to avoid the problem of replacement values is to use a PDAE-Encoder approach similar to what is done in the regular imputeLM model. If you learn a model to encode similar fully-observed and partially-observed samples to a similar latent representation, such a missingness-robust encoding could provide a useful set of conditioning information. One could then concatenate the encoded version of the reference observation to the noise for G and to the generated/real mask for D, based on the idea that the encoding is sufficiently abstract such that the discriminator neither recognizes the missingness in it nor recognizes the specific observation. It would then require careful tuning of the size of the latent dimension since a large size might not be robust enough, but a small size could lose a substantial amount of information about the variables, which would then impede G and D from learning the MAR and MNAR information.

G Hyperparameters and Neural Network Architecture

The total runtime of the experiments for which the results are listed in this study is approximately 55 hours on a notebook GPU. Therefore additional hyperparameter optimization and more experiments were not feasible.

All of the choices for the neural network architecture are a combination of empirical exploration of different options and taking the results of similar works as a guide - Pereira et al. (2020) give an overview of common practices. All models except the VAE use ReLU activation functions. The VAE uses a Leaky ReLU function with a negative slope of 0.0001. The PDAE-Encoder model has two hidden layers with 2000 and 700 neurons, respectively. It is trained with uniform-replacement noise randomly applied to a proportion of the features, which is determined by the missingness level in the data. The MP model has three hidden layers of 2000 neurons each. After each hidden layer, there is a dropout layer with a dropout probability of 0.5 to aid generalization capabilities. Its last layer is followed by a sigmoid nonlinearity. The ADAE model has six hidden layers of 2000 neurons each. In the corruption process of the ADAE model, we use the MP model-generated masks to corrupt the observations by setting the masked variables to zero. The benchmark DAE model has four hidden layers with 2000 neurons each. The benchmark VAE encoder and decoder each have one hidden layer of 2000 neurons and a layer corresponding to the latent dimension of 500. For the experiments in Section 4.4, we set the size of the last PDAE-Encoder layer to 50 to examine the effect of the Encoder in a more pronounced scenario.

We train all models for a maximum of 10 epochs with possible early stopping if the out-of-sample loss does not decrease any more. The learning rate is set to $3e-4$ without decay for all models. The Adam optimizer by Kingma and Ba (2014) is used with $\beta = (0.9, 0.999)$ and $\varepsilon = 1e-8$ for optimization of the neural network parameters.

The parameters of the missingness generation algorithms are as follows: The size of each patch in PATCHES is 5x5 pixels. For the generation of p_j in MAR and MNAR, we add $1e-6$

times the identity matrix to the estimated covariance matrix to make it positive definite. For the sigmoid function applied to the samples from the multivariate normal, the offset is 0.05 for a missingness level of 0.2 and 0.15 for a level of 0.1. The sigmoid scaling factor is always 10. For QMNAR, we assign 20% of the variables to have missing values in their highest and lowest 20% of values with a probability of 0.8.

H Implementation

All neural network components are implemented in PyTorch (Paszke et al., 2019). For the Variational Autoencoder, the Generative Adversarial Network and the downstream classification model code from Raschka (2019) were adapted. For the computation of the QMNAR missingness, we rely on the Python implementation by Muzellec (2019). The tutorial code on the PyTorch website was used as a starting point for the neural network implementations.

In Table G.1, we list all the Python (Van Rossum and Drake, 2009) packages that were used together with their purpose.

Table H.1: A table that indicates which python package was used in the implementation for which functionality.

Task	**Python Package**	**Reference**
Neural Network Models	PyTorch	Paszke et al. (2019)
GPU Acceleration	CUDA	NVIDIA et al. (2020)
Model logging	TensorBoard, Weights and Biases	Abadi et al. (2016), Biewald (2020)
Image Data Handling	TorchVision	TorchVision maintainers (2016)
Mean Imputation, Preprocessing	sklearn	Pedregosa et al. (2011)
Line Search	scipy	Virtanen et al. (2020)
Basic operations	numpy, pandas	Harris et al. (2020), McKinney (2010)
Visualizations	matplotlib	Hunter (2007)

The full implementation of the model proposed here, together with scripts to replicate the experiments, can be found in this GitHub repository. The repository is structured in the following way. It has one main notebook `imputation_models.ipynb`. This notebook brings together all parts of dataset generation, model training, testing and visualization, making use of all the `helper_*.py` files. This notebook can be used to do manual experiments with the model. It contains the implementation of the imputeLM model as well as the DAE, VAE and Mean imputation benchmark models. At the top of the notebook, a dictionary is included, which specifies all the data parameters (which dataset and which missingness parameters to use). Then for each model, there are another two dictionaries that specify the model configuration parameters and the model training. The MP model is implemented with the option to use a VAE or GAN instead of the vanilla neural network that is used in this study.

There is one file for each subsection of the results section to reproduce the experiments. These scripts are named with `experiment_*` in front. They produce a set of CSV files for each seed separately. Additionally, they produce an Excel file that contains the averages and standard deviations, if applicable. The `experiment_mask_prediction.ipynb` produces its table and plots inside the notebook.

In the code, the ADAE model is mostly referred to as the SDAE model, and noise and missingness are often used interchangeably. The benchmark DAE model is referred to as BDAE and the PDAE-Encoder as Encoder.