WHY NOT TO USE ZERO IMPUTATION? CORRECTING SPARSITY BIAS IN TRAINING NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Handling missing data is one of the most fundamental problems in machine learning. Among many approaches, the simplest and most intuitive way is zero imputation, which treats the value of a missing entry simply as zero. However, many studies have experimentally confirmed that zero imputation results in suboptimal performances in training neural networks. Yet, none of the existing work has explained what brings such performance degradations. In this paper, we introduce the *variable sparsity problem (VSP)*, which describes a phenomenon where the output of a predictive model largely varies with respect to the rate of missingness in the given input, and show that it adversarially affects the model performance. We first theoretically analyze this phenomenon and propose a simple yet effective technique to handle missingness, which we refer to as *Sparsity Normalization (SN)*, that directly targets and resolves the VSP. We further experimentally validate SN on diverse benchmark datasets, to show that debiasing the effect of input-level sparsity improves the performance and stabilizes the training of neural networks.

1 INTRODUCTION

Many real-world datasets often contain data instances whose subset of input features is missing. While various imputing techniques, from imputing using global statistics such as mean, to individually imputing by learning auxiliary models such as GAN, can be applied with their own pros and cons, the most simple and natural way to do this is *zero imputation*, where we simply treat a missing feature as zero. In neural networks, at first glance, zero imputation can be thought of as a reasonable solution since it simply drops missing input nodes by preventing the weights associated with them from being updated. Some what surprisingly, however, many previous studies have reported that this intuitive approach has an adverse effect on model performances (Hazan et al., 2015; Luo et al., 2018; Śmieja et al., 2018), and none of them has investigated the reasons of such performance degradations.

In this work, we find that zero imputation causes the output of neural network to largely vary with respect to the number of missing entries in the input. We name this phenomenon *Variable Sparsity Problem (VSP)*, which should be avoided in many real-world tasks. Consider a movie recommender system, for instance. It is not desirable that users get different average of predicted ratings just because they have rated different number of movies (regardless of their actual rating values). One might argue that people with less ratings do not like movies in general and it is natural to give higher predicted values to people with more ratings. This might be partially true for users of *some* sparsity levels, but it is not a common case uniformly applicable for a wider range of sparsity levels. This can be verified in real collaborative filtering datasets as shown in Figure 1 (upper left corner) where users have a similar average rating for test data regardless of the number of known ratings (see also other two examples in Figure 1). However, in standard neural networks with zero imputation, we observe that the model's inference correlates with the number of known entries of the data instance as shown in the second row of Figure 1[1]. It would be fatal in some safety-critical applications such as a medical

[1]Note that, this tendency is very consistent with other test points and is observed throughout the entire learning process (even before the training).

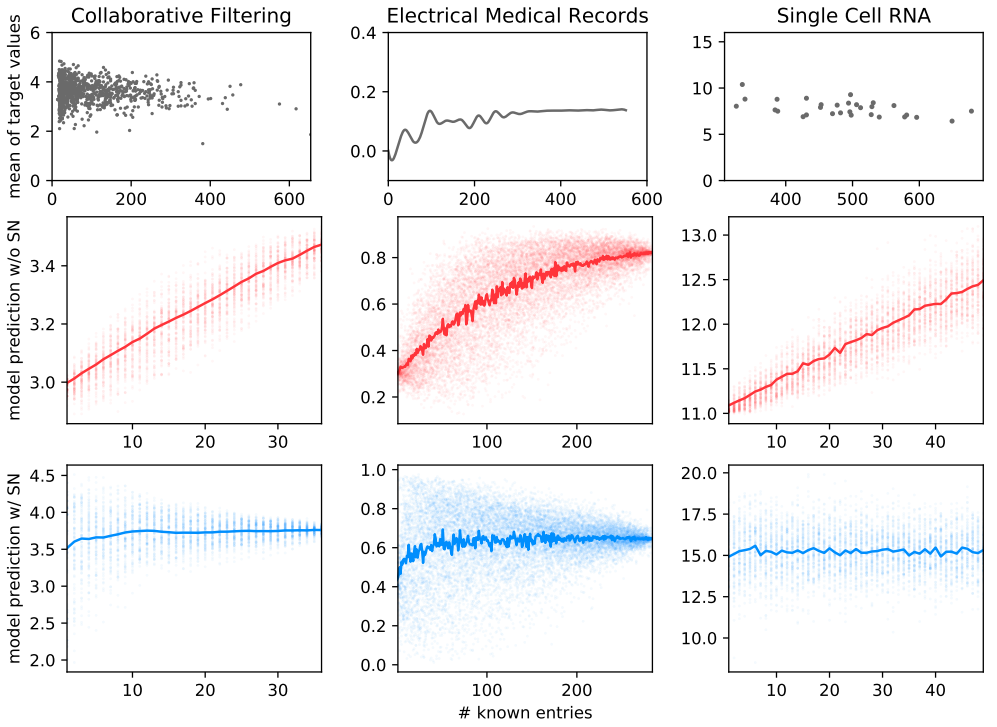


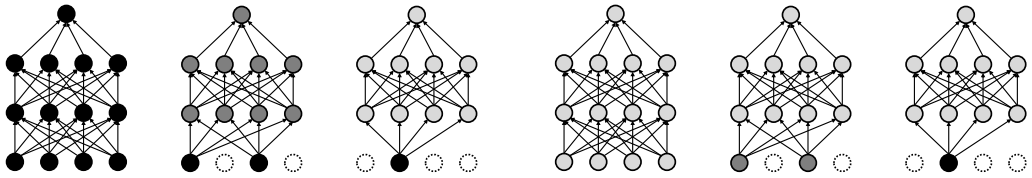
Figure 1: **First column**: Sedhain et al. (2015) on Movielens 100K (collaborative filtering) dataset. **Second column**: LSTM on Physionet 2012 (electrical medical records) dataset. **Third column**: Talwar et al. (2018) on Blakeley (single cell RNA sequence) dataset. **First row**: Mean of target values according to the number of known entries in training set. **Second row**: Predicted values of models with zero imputation according to the number of known entries for a randomly selected test point. Input masks are randomly sampled (to artificially control its sparsity level). For each target sparsity level through x-axis, 50 samples are drawn, scattering the predicted values and plotting the average in solid line. **Third row**: Figures on how plots in second row are corrected by Sparsity Normalization.

domain: a patient's probability of developing disease for example should not be evaluated differently depending on the number of medical tests they received (we do not want our model to predict the probability of death is high just because some patient has been screened a lot!).

In addition, we theoretically analyze the existence of VSP under several circumstances and propose a simple yet effective means to suppress VSP while retaining the intuitive advantages of zero imputation: normalizing with the number of non-zero entries for each data instance. We refer to this regularization as *Sparsity Normalization*, and show that it effectively deals away with the VSP, resulting in significant improvements in both the performance and the stability of training neural networks.

Our contribution in this paper is threefold:

- To best of our knowledge, we are the first in exploring the adverse effect of zero imputation, both theoretically and empirically.
- We identify the cause of adverse effect of zero imputation, which we refer to as variable sparsity problem, and formally describe how this problem actually affects training and inference of neural networks (Section 2). We further provide new perspectives using VSP to understand phenomena that have not been clearly explained or that we have misunderstood (Section 4 and 5).
- We present Sparsity Normalization (SN) and theoretically show that SN can solve the VSP under certain conditions (Section 3). We also experimentally reaffirm that simply applying SN can effectively alleviate or solve the VSP yielding significant performance gains (Section 4).



(a) Neural network with variable sparsity levels (b) with Sparsity Normalization

Figure 2: The change of output values according to the sparsity level of inputs (darker color indicates greater absolute value and dotted circles indicate missing nodes with zero imputation). SN makes the possible output range of a network be more stable with respect to sparsity level.

2 VARIABLE SPARSITY PROBLEM

We formally define the *Variable Sparsity Problem (VSP)* as follows: a phenomenon in which the expected value of the output layer of a neural network (over the weight and input distributions) depends on the sparsity (the number of zero values) of the input data (Figure 2a). With VSP, the activation values of neural networks could become largely different for exactly the same input instance, depending on the number of zero entries; this makes training more difficult and may mislead the model into incorrect predictions.

While zero imputation is intuitive in the sense that it drops the missing input features, we will show that it causes this VSP problem for several example cases. Specifically, we show the VSP under assumptions with increasing generality: **(Case 1)** where activation function is an identity mapping with no bias, **(Case 2)** where activation function is an affine function, and **(Case 3)** where activation function is a non-decreasing convex function such as ReLU (Glorot et al., 2011), leaky ReLU (Maas et al., 2013), ELU (Clevert et al., 2016), or Softplus (Dugas et al., 2001).

Here, we summarize the notation for clarity. For a L-layer deep network with non-linearity σ, we use $W^i \in \mathbb{R}^{n_i \times n_{i-1}}$ to denote the weight matrix of i-th layer, $\mathbf{b}^i \in \mathbb{R}^{n_i}$ to denote the bias, $\mathbf{h}^i \in \mathbb{R}^{n_i}$ to denote the activation vector. For simplicity, we use $\mathbf{h}^0 \in \mathbb{R}^{n_0}$ and $\mathbf{h}^L \in \mathbb{R}^{n_L}$ to denote input and output layer, respectively. Then, we have

$$\mathbf{h}^i = \sigma(W^i \mathbf{h}^{i-1} + \mathbf{b}^i), \quad \text{for } i = 1, \cdots, L.$$

Our goal in this section is to observe the change in $\mathbf{h}^L$ as the sparsity of $\mathbf{h}^0$ (input $\mathbf{x}$) changes. To simplify the discussion, we consider the following assumption:

Assumption 1. *(i) Every coordinate of input vector, h_l^0, is generated by the element-wise multiplication of two random variables $\tilde{h}_l^0$ and m_l where m_l is binary mask indicating missing value and $\tilde{h}_l^0$ is a (possibly unobserved) feature value. Here, missing mask m_l is MCAR (missing completely at random), with no dependency with other mask variables or their values $\tilde{\mathbf{h}}^0$. All m_l follow some identical distribution with mean μ_m. (ii) The elements of matrix W^i are mutually independent and follow the identical distribution with mean μ_w^i. Similarly, $\mathbf{b}^i$ and $\tilde{\mathbf{h}}^0$ consist of i.i.d. coordinates with mean μ_b^i and μ_x, respectively. (iii) μ_w^i is not zero uniformly over all i.*

(i) assumes the simplest missing mechanism. (ii) is similarly defined in Glorot & Bengio (2010) and He et al. (2015) in studying weight initialization techniques. (iii) may not hold under some initialization strategies, but as the learning progresses, it is very likely to hold.

(Case 1) For simplicity, let us first consider networks without the non-linearity nor the bias term. Theorem 1 shows that the average value of the output layer $E[h_l^L]$ is directly proportional to the expectation of the mask vector μ_m:

Theorem 1. *Suppose that activation σ is an identity function and that b_l^i is uniformly fixed as zero under Assumption 1. Then, we have $E[h_l^L] = \prod_{i=1}^{L} n_{i-1} \mu_w^i \mu_x \mu_m$.*

(Case 2) When the activation function is affine but now with a possibly nonzero bias, $E[h_l^L]$ is influenced by μ_m in the following way:

Theorem 2. *Suppose that activation σ is an affine function under Assumption 1. Suppose further that $f_i(x)$ is defined as $\sigma(n_{i-1}\mu_w^i x + \mu_b^i)$. Then, $E[h_l^L] = f_L \circ \cdots \circ f_1(\mu_x \mu_m)$.*

(Case 3) Finally, when the activation function is non-linear but non-decreasing and convex, we can show that $E[h_l^L]$ is lower-bounded by some quantity involving μ_m:

Theorem 3. *Suppose that σ is a non-decreasing convex function under Assumption 1. Suppose further that $f_i(x)$ is defined as $\sigma(n_{i-1}\mu_w^i x + \mu_b^i)$. Then, $E[h_l^L] \geq f_L \circ \cdots \circ f_1(\mu_x \mu_m)$.*

If the expected value of the output layer (or the lower bound of it) depends on the level of sparsity/missingness as in Theorem 1-3, even similar data instances may have different output values depending on their sparsity levels, which would hinder fair and correct inference of the model. As shown in Figure 1 (second row), the VSP can easily occur even in practical settings of training neural networks where the above conditions do not hold.

3 Sparsity Normalization

In this section, we propose a simple yet surprisingly effective method to resolve the VSP. We first revisit Case 2 to find a way of making expected output independent of input sparsity level since the linearity in activation simplifies the correction. Recalling the notation of $\mathbf{h}^0 = \tilde{\mathbf{h}}^0 \odot \mathbf{m}$ ($\odot$ represents the element-wise product), we find that simply normalizing via $\mathbf{h}^0_{\text{SN}} = (\tilde{\mathbf{h}}^0 \odot \mathbf{m}) \cdot K_1/\mu_m$ for any fixed constant K_1, can debias the dependency on the input sparsity level. We name this simple normalizing technique *Sparsity Normalization* (SN) and describe it in Algorithm 1. Conceptually, this method scales the size of each input value according to its sparsity level so that the change in output size is less sensitive to the sparsity level (Figure 2b). The formal description on correcting sparsity bias by SN is as follows in this particular case:

Algorithm 1 Sparsity Normalization (SN)

Input: Dataset $\mathcal{D}$, constant K.
Output: Sparsity Normalized Dataset $\mathcal{D}_{\text{SN}}$.

Empty set $\mathcal{S} = \phi$
for each $(\mathbf{h}^0, \mathbf{m}) \in \mathcal{D}$ **do**
  $\mathbf{h}^0_{\text{SN}} \leftarrow K \cdot \mathbf{h}^0 / \|\mathbf{m}\|_1$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{h}^0_{\text{SN}}\}$
end for
$\mathcal{D}_{\text{SN}} \leftarrow \mathcal{S}$

Theorem 4. *(With Sparsity Normalization) Suppose that activation σ is an affine function under Assumption 1. Suppose further that $f_i(x) = \sigma(n_{i-1}\mu_w^i x + \mu_b^i)$ and replace the input layer using SN, i.e. $\mathbf{h}^0_{SN} = (\tilde{\mathbf{h}}^0 \odot \mathbf{m}) \cdot K_1/\mu_m$ for any fixed constant K_1. Then, we have $E[h_l^L] = f_L \circ \cdots \circ f_1(\mu_x \cdot K_1)$.*

Unlike in Theorem 2, SN in Theorem 4 makes average activation to be independent of μ_m, which determines the sparsity levels of input. It is not trivial to show the counterpart of **(Case 3)** using SN since $E[\sigma(x)] = \sigma(E[x])$ does not hold in general. However, we show through extensive experiments in Section 4 that SN is practically effective even in more general cases.

While Theorem 4 assumes that μ_m is known and fixed across all data instances, we relax this assumption in practice and consider varying μ_m across data instances. Specifically, by a maximum likelihood principle, we can estimate μ_m for each instance by $\left\|\mathbf{h}^0\right\|_0 / n_0 = \|\mathbf{m}\|_1 / n_0$. Thus, we have $\mathbf{h}^0_{\text{SN}} = K \cdot \mathbf{h}^0 / \|\mathbf{m}\|_1$ where $K = n_0 \cdot K_1$ (see Algorithm 1). In practice, we recommend using K as the average of $\|\mathbf{m}\|_1$ over all instances in the training set. We could encounter the dying ReLU phenomenon (He et al., 2015) if K is too small (e.g. $K = 1$). Since the hyper-parameter K can bring in a regularization effect via controlling the magnitude of gradient (Salimans & Kingma, 2016), we define $K = \mathbf{E}_{(\mathbf{h}^0, \mathbf{m}) \in \mathcal{D}}[\|\mathbf{m}\|_1]$ so that the average scales remain constant before and after the normalization, minimizing such side effects caused by SN.

Table 1: Debiasing variable sparsity using SN on Movielens datasets. Test RMSE with 95% confidence interval of 5-runs is provided.

Datasets			Movielens 100K	Movielens 1M	Movielens 10M
user vector	AutoRec (Sedhain et al., 2015)	w/o SN	0.9346 ± 0.0007	0.8831 ± 0.0002	0.8859 ± 0.0014
		w/ SN	$\mathbf{0.9208 \pm 0.0023}$	$\mathbf{0.8742 \pm 0.0003}$	$\mathbf{0.8462 \pm 0.0005}$
	CF-NADE (Zheng et al., 2016)	w/o SN	0.9253 ± 0.0010	0.8530 ± 0.0006	0.8113 ± 0.0058
		w/ SN	$\mathbf{0.9231 \pm 0.0012}$	$\mathbf{0.8525 \pm 0.0006}$	$\mathbf{0.7854 \pm 0.0006}$
item vector	AutoRec (Sedhain et al., 2015)	w/o SN	0.8835 ± 0.0003	0.8320 ± 0.0003	0.7807 ± 0.0017
		w/ SN	$\mathbf{0.8809 \pm 0.0011}$	$\mathbf{0.8294 \pm 0.0004}$	$\mathbf{0.7706 \pm 0.0023}$
	CF-NADE (Zheng et al., 2016)	w/o SN	0.8982 ± 0.0005	0.8405 ± 0.0007	N/A
		w/ SN	$\mathbf{0.8900 \pm 0.0018}$	$\mathbf{0.8366 \pm 0.0008}$	N/A

Table 2: Comparison of AutoRec with SN against seven state-of-the-art collaborative filtering methods for each datasets. Bold fonts indicate neural network based models. The results marked † are taken from Zhang et al. (2017) and all other baselines are taken from original papers.

Models	Movielens 100K	Models	Movielens 1M	Models	Movielens 10M
Koren (2008)	$0.913^{\dagger}$	**Dziugaite & Roy (2015)**	0.843	**Sedhain et al. (2015)**	0.782
Zhuang et al. (2017)	0.9114 ± 0.0093	Fu et al. (2018)	0.836	**Berg et al. (2018)**	0.777
Koren et al. (2009)	$0.911^{\dagger}$	Lee et al. (2016)	0.8333	**Zheng et al. (2016)**	0.771
Dziugaite & Roy (2015)	0.903	Yi et al. (2019)	0.8321	Li et al. (2016)	0.7682 ± 0.0003
Zhang et al. (2017)	0.901	**Berg et al. (2018)**	0.832	Chen et al. (2017)	0.7672 ± 0.0001
Yi et al. (2019)	0.8889	**Sedhain et al. (2015)**	0.831	Fu et al. (2018)	0.766
Lee et al. (2016)	0.8881 ± 0.0017	**Zheng et al. (2016)**	0.829	Li et al. (2017)	0.7634 ± 0.0002
AutoRec w/ SN (ours.)	0.8816 ± 0.0087	**AutoRec w/ SN (ours.)**	0.8260 ± 0.0023	**AutoRec w/ SN (ours.)**	0.7690 ± 0.0023

4 EXPERIMENTS

In this section, we empirically show that VSP occurs in various machine learning tasks and it can be alleviated by SN. In addition, we also show that resolving VSP leads to improved model performance on diverse scenarios.

4.1 COLLABORATIVE FILTERING (RECOMMENDATION) DATASETS

We identify VSP and the effect of SN on several popular benchmark datasets for collaborative filtering with extremely high missing rates. We train a AutoRec (Sedhain et al., 2015) using user vector on Movielens (Harper & Konstan, 2016) 100K dataset for validating VSP and SN. Going back to the first column of Figure 1, the prediction with SN is almost constant regardless of $\|\mathbf{m}\|_1$. Another phenomenon that we observe in Figure 1 is that the higher the $\|\mathbf{m}\|_1$, the smaller the variation in the prediction of the model with SN. Note that the same tendency has been observed regardless of test instances or datasets. This implies that models with SN yield more calibrated predictions; as more features are known for a particular instance, the variance of prediction for that instance should decrease (since we generated independent masks in Figure 1). It is also worthwhile to note that AutoRec is a sigmoid-based network and Movielens datasets are known to have no MCAR hypothesis (Wang et al., 2018), in which Assumption 1 does not hold at all.

We also validate that we can obtain performance gains by simply applying SN to AutoRec (Sedhain et al., 2015) and CF-NADE (Zheng et al., 2016), which are the states-of-the-arts among neural networks based collaborative filtering models on several Movielens datasets (see Appendix B for detailed settings). In Table 1[2], we consider three different sized Movielens datasets, on each of which we evaluate two data encoding (user- or item-rating vector based) methods. While we obtain the performance improvement by applying SN in all cases, it is more prominent in user-rating based model.

[2]We consider a CF-NADE without weight sharing and re-run experiments for fair comparisons because applying SN with weight sharing is not trivial. We also excluded averaging possible choices because it does not make big differences given unnecessary extra computational costs.

Table 3: Debiasing variable sparsity using SN on five disease identification tasks of NHIS dataset. Test AUROC with 95% confidence interval of 5-runs is provided.

Dataset	Cardiovascular	Fatty Liver	Hypertension	Heart Failure	Diabetes
w/o SN	0.7057 ± 0.0027	0.6750 ± 0.0050	0.7977 ± 0.0027	0.7834 ± 0.0036	0.9121 ± 0.0097
w/ SN	**0.7106 ± 0.0005**	**0.6911 ± 0.0022**	**0.8096 ± 0.0010**	**0.7914 ± 0.0012**	**0.9283 ± 0.0011**

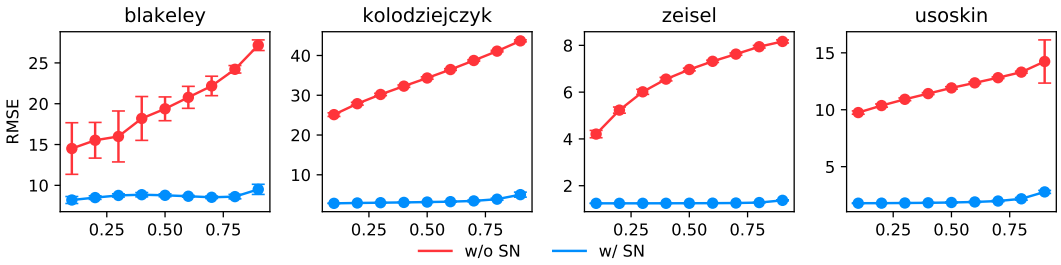


Figure 3: Debiasing variable sparsity using SN according to test set ratio on four imputation tasks of single cell RNA sequence dataset. Test RMSE with 95% confidence interval of 10-runs is provided.

Furthermore, Table 2 compares our simply modification using SN on AutoRec with other states-of-the-art collaborative filtering models beyond neural networks[3]. Unlike experiments in Table 1, which uses the same network architectures proposed in original papers, here we could successfully learn more expressive network due to the stability obtained by using SN. It is important to note that all models outperforming AutoRec with SN on Movielens 10M are ensemble models while AutoRec with SN is a single model and it shows consistently competitive results across all datasets.

4.2 Electric Medical Records (EMR) Datasets

We further test SN for clnical time-series prediction with two Electric Medical Records (EMR), namely PhysioNet Challenge 2012 (Silva et al., 2012) and the National Health Insurance Service (NHIS) datasets, which have intrinsic missingness since patients will only receive medical examinations that are considered to be necessary. We identify whether the VSP exists with the PhysioNet Challenge 2012 dataset (Silva et al., 2012). We randomly select one test point and plot in-hospital death probabilities as the number of examinations varies (Second column of Figure 1). Without SN, the in-hospital death probability increases as the number of examinations increases, even though there is no such tendency in the dataset statistics. However, SN corrects this bias so that in-hospital death probability is consistent regardless of the number of examinations. We observe a similar tendency for examples from the NHIS dataset as well.

Although SN corrects the VSP in both datasets, we observe different behaviors in both cases in terms of actual performance changes. While SN significantly outperforms its counterpart without SN on NHIS dataset as shown in Table 3, it just performs similarly on PhysioNet dataset (results and detailed settings are deferred to Appendix C). However, SN is still valuable for its ability to prevent biased predictions in this mission-critical area.

4.3 Single-cell RNA Sequence Datasets

Single-cell RNA sequence datasets contain expression levels of specific RNA for each single cells. AutoImpute (Talwar et al., 2018) is one of the states-of-the-art methods that imputes missing data on single-cell RNA sequence datasets. We reproduce their experiments using authors' official implementation, and follow most of their experimental settings (see Appendix D for details).

[3]Overfitting is less with SN. This is why we use twice the capacity than the existing AutoRec model.

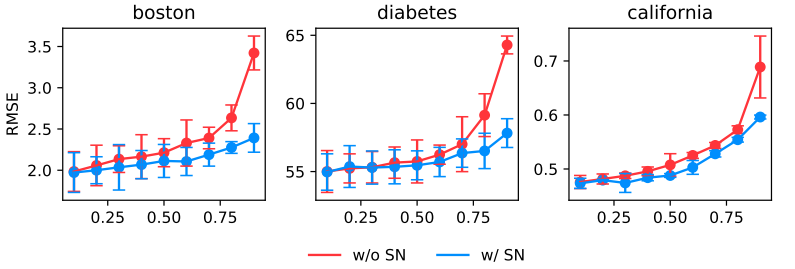


Figure 4: Debiasing variable sparsity using SN with respect to drop rates on three popular UCI regression datasets. Test RMSE with 95% confidence interval of 5-runs is provided.

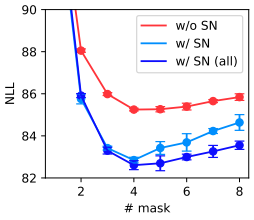


Figure 5: Negative log likelihood of MADE on binarized MNIST with and without SN

As before, we first check whether VSP occurs in AutoImpute model. The third column in Figure 1 shows how the prediction of AutoImpute model changes as the number of known entries changes. Although the number of RNAs found in the specific cell is less related to cell characteristics (upper right corner in Figure 1), the prediction increases as the number of RNAs found in the cell increases. This tendency is dramatically reduced with SN.

Figure 3 shows how imputation performance changes by applying SN to several single cell RNA sequence datasets with respect to the portion of train set. (see Appendix D for more results.) As we can see in Figure 3, SN significantly increases the imputation performance of AutoImpute model. In particular, the smaller the train data, the better the SN effect in all datasets consistently. AutoImpute model is a sigmoid-based function and single cell RNA datasets (Talwar et al., 2018) do not have the MCAR hypothesis, unlike Assumption 1. Nevertheless, VSP occurs even here and it can be successfully alleviated by SN with huge performance gain. It implies that SN would work for other neural networks based imputation techniques.

4.4 Dropout on UCI datasets

While SN primarily targets to fix the VSP in the input layer, it could be also applied to any layers of deep neural networks to resolve VSP. The typical example of having heterogeneous sparsity in hidden layers is when we use dropout (Srivastava et al., 2014), which can be understood as another form of zero imputation but at the hidden layers; with Bernoulli dropout, the variance of the number of zero units across instances is $np(1-p)$ (n: the dimension of hidden layer, p: drop rate). While dropout partially handles this VSP issue by scaling $1/(1-p)$ in the training phase[4], SN can *exactly* correct VSP of hidden layers by considering individual level sparsity (Note that the scaling of dropout can be viewed as applying SN in an average sense: $\mathbb{E}\left[\|\mathbf{m}\|_1\right] = n(1-p)$ and $K = n$).

Figure 4 shows how the RMSE changes as the drop rate changes with and without SN on three popular UCI regression datasets (Boston Housing, Diabetes, and California Housing)[5]. Figure 4 shows that the larger the drop rate, the greater the difference of RMSE between with and without SN. To explain this phenomenon, we define the degree of VSP as the inverse of *signal-to-noise ratio* with respect to the number of active units in hidden layers: $\sqrt{p/(n(1-p))}$ (expected number of active units over its standard deviation). As can be seen from the figure, the larger drop rate p is, the more severe degree of VSP is and thus the greater protection by SN against performance degradation.

[4]In almost all of the deep learning frameworks such as PyTorch, TensorFlow, Theano and Caffe, this inverted dropout is supported.

[5]The most experimental settings are adopted from Klambauer et al. (2017); Littwin & Wolf (2018)'s UCI experiments (see Appendix E for detail).

4.5 DENSITY ESTIMATION

In the current literature of estimating density based on deep models, inputs with missing features are in general not largely considered. However, we still may experience the VSP since the proportion of zeros in the data itself can vary greatly from instance to instance. In this experiment, we apply SN to MADE (Germain et al., 2015), which is the one of modern architectures in neural network-based density estimation. We reproduce binarized MNIST (LeCun, 1998) experiments of Germain et al. (2015) measuring negative log likelihood (the lower the better) of test dataset while increasing the number of masks. Figure 5 illustrates the effect of using SN. Note that MADE uses masks in the hidden layer that are designed to produce proper autoregressive conditional probabilities and variable sparsity arises across hidden nodes. SN can be trivially extended to handle this case as well and the corresponding result is given in the figure (denoted as `w/SN(all)`). We reaffirm that SN is effective even when MCAR assumption is not established.

5 RELATED WORKS

Missing handling techniques Missing imputation can be understood as a technique to increase the generalization performance by injecting plausible noise into data. Noise injection using global statistics like mean or median values is the simplest way to do this (Lipton et al., 2016; Śmieja et al., 2018). However, it could lead to highly incorrect estimation since they do not take into consideration the characteristics of each data instance (Tresp et al., 1994; Che et al., 2018). To overcome this limitation, researchers have proposed various ways to model individualized noise using autoencoders, and GANs (Pathak et al., 2016; Yoon et al., 2018). However, those model based imputation techniques have not properly worked for high dimensional datasets with the large number of features and/or extremely high missing rates (Yoon et al., 2018) because excessive noise can ruin the training of neural networks rather increasing generalization performance.

For this reason, in the case of high dimensional datasets such as collaborative filtering or single cell RNA sequences, different methods of handling missing data have been proposed. A line of work simply used zero imputation by minimizing noise level and achieved states-of-the-art performance on their target datasets (Sedhain et al., 2015; Zheng et al., 2016; Talwar et al., 2018). In addition, methods using low-rank matrix factorization have been proposed to reduce the input dimension, but these methods not only cause lots of information loss but also fail to capture non-linearity of the input data (Hazan et al., 2015; Bachman et al., 2017; He et al., 2017). Vinyals et al. (2016); Monti et al. (2017) proposed recurrent neural networks (RNN) based methods but computational costs for these methods are excessive for high dimensional datasets. Also, it is not natural to use RNN-based models for non-sequential datasets.

Other forms of Sparsity Normalization We discuss other forms of SN to alleviate VSP, already in use unwittingly due to empirical performance improvements. DropBlock (Ghiasi et al., 2018) compensates activation from dropped features by exactly counting mask vector similar to SN (similar approach discussed in Section 4.4. It is remarkable that we can find models using SN-like normalization even in handling datasets without missing features. For example, in CBOW model (Mikolov et al., 2013) where the number of words used as an input depends on the position in the sentence, it was later revealed that SN like normalization has a practical performance improvement. As an another example, Kipf & Welling (2017) applied Laplacian normalization which is the standard way of representing a graph in graph theory, can naturally handle heterogeneous node degrees and precisely matches the SN operation. In this paper, we explicitly extend SN, which was limited and unintentionally applied to only few settings, to a model agnostic technique.

6 CONCLUSION

We identified *variable sparsity problem (VSP)* caused by zero imputation that has not been explicitly studied before. To best of our knowledge, this paper provided the first theoretical analysis on why zero imputation is harmful to inference of neural networks. We showed that variable sparsity problem actually exists in diverse real-world datasets. We also confirmed that theoretically inspired normalizing method, Sparsity Normalization, not only reduces the VSP but also improves the generalization performance and stability of feed-forwarding of neural networks with missing values, even in areas where existing missing imputation techniques do not cover well (e.g., collaborative filtering, single cell RNA datasets).

REFERENCES

Philip Bachman, Alessandro Sordoni, and Adam Trischler. Learning algorithms for active learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 301–310. JMLR. org, 2017.

Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining Deep Learning Day*. ACM, 2018.

Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):6085, 2018.

Chao Chen, Dongsheng Li, Qin Lv, Junchi Yan, Li Shang, and Stephen M Chu. Gloma: Embedding global information in local matrix approximation models for collaborative filtering. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *International Conference on Learning Representations*, 2016.

Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. In *Advances in neural information processing systems*, pp. 472–478, 2001.

Gintare Karolina Dziugaite and Daniel M Roy. Neural network matrix factorization. *arXiv preprint arXiv:1511.06443*, 2015.

Mingsheng Fu, Hong Qu, Dagmawi Moges, and Li Lu. Attention based collaborative filtering. *Neurocomputing*, 311:88–98, 2018.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pp. 881–889, 2015.

Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, pp. 10727–10737, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.

F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):19, 2016.

Elad Hazan, Roi Livni, and Yishay Mansour. Classification with low rank and missing data. In *Proceedings of the 32nd International Conference on Machine Learning-Volume 68*, pp. 257–266. JMLR. org, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pp. 173–182. International World Wide Web Conferences Steering Committee, 2017.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pp. 971–980, 2017.

Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 426–434. ACM, 2008.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, pp. 30–37, 2009.

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

Joonseok Lee, Seungyeon Kim, Guy Lebanon, Yoram Singer, and Samy Bengio. Llorma: Local low-rank matrix approximation. *The Journal of Machine Learning Research*, 17(1):442–465, 2016.

Dongsheng Li, Chao Chen, Qin Lv, Junchi Yan, Li Shang, and Stephen Chu. Low-rank matrix approximation with stability. In *International Conference on Machine Learning*, pp. 295–303, 2016.

Dongsheng Li, Chao Chen, Wei Liu, Tun Lu, Ning Gu, and Stephen Chu. Mixture-rank matrix approximation for collaborative filtering. In *Advances in Neural Information Processing Systems*, pp. 477–485, 2017.

Zachary C Lipton, David C Kale, and Randall Wetzel. Modeling missing data in clinical time series with rnns. *Machine Learning for Healthcare*, 2016.

Etai Littwin and Lior Wolf. Regularizing by the variance of the activations' sample-variances. In *Advances in Neural Information Processing Systems*, pp. 2115–2125, 2018.

Yonghong Luo, Xiangrui Cai, Ying Zhang, Jun Xu, et al. Multivariate time series imputation with generative adversarial networks. In *Advances in Neural Information Processing Systems*, pp. 1596–1607, 2018.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, pp. 3, 2013.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*, 2013.

Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 3697–3707, 2017.

Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2536–2544, 2016.

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 901–909, 2016.

Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 111–112. ACM, 2015.

I. Silva, George B. Moody, Daniel J. Scott, L. A. Celi, and R. Gritz Mark. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. *2012 Computing in Cardiology*, pp. 245–248, 2012.

Marek Śmieja, Łukasz Struski, Jacek Tabor, Bartosz Zieliński, and Przemysław Spurek. Processing of missing data by neural networks. In *Advances in Neural Information Processing Systems*, pp. 2719–2729, 2018.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Divyanshu Talwar, Aanchal Mongia, Debarka Sengupta, and Angshul Majumdar. Autoimpute: Autoencoder based imputation of single-cell rna-seq data. *Scientific reports*, 8(1):16329, 2018.

Volker Tresp, Subutai Ahmad, and Ralph Neuneier. Training neural networks with deficient data. In *Advances in neural information processing systems*, pp. 128–135, 1994.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. In *International Conference on Learning Representations*, 2016.

Menghan Wang, Mingming Gong, Xiaolin Zheng, and Kun Zhang. Modeling dynamic missingness of implicit feedback for recommendation. In *Advances in neural information processing systems*, pp. 6669–6678, 2018.

Baolin Yi, Xiaoxuan Shen, Hai Liu, Zhaoli Zhang, Wei Zhang, Sannyuya Liu, and Naixue Xiong. Deep matrix factorization with implicit feedback embedding for recommendation system. *IEEE Transactions on Industrial Informatics*, 2019.

Jinsung Yoon, James Jordon, and Mihaela Van Der Schaar. Gain: Missing data imputation using generative adversarial nets. In *Proceedings of the 35th International Conference on Machine Learning-Volume 71*, 2018.

Shuai Zhang, Lina Yao, and Xiwei Xu. Autosvd++: An efficient hybrid collaborative filtering model via contractive auto-encoders. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 957–960. ACM, 2017.

Yin Zheng, Bangsheng Tang, Wenkui Ding, and Hanning Zhou. A neural autoregressive approach to collaborative filtering. In *Proceedings of the 33rd International Conference on Machine Learning-Volume 69*. JMLR. org, 2016.

Fuzhen Zhuang, Zhiqiang Zhang, Mingda Qian, Chuan Shi, Xing Xie, and Qing He. Representation learning via dual-autoencoder for recommendation. *Neural Networks*, 90:83–89, 2017.

A PROOFS

A.1 PROOF OF THEOREM 1

Proof. From the definition of $h_l, w_l^1, h_l^0, \tilde{h}_l^0, m_l$, the following equation holds.

$$E[h_l^1] = n_0 E[w_l^1 h_1^0] = n_0 E[w_l^1 \tilde{h}_l^0 m_l]$$

From the Assumption 1, w_l^1, $\tilde{h}_l^0$, and m_l are independent of each other. Thus,

$$E[h_l^1] = n_0 E[w_l^1] E[\tilde{h}_l^0] E[m_l]$$

Similarly, the following holds.

$$E[h_l^i] = n_{i-1} E[w_l^i h_l^{i-1}] \text{ for } i = 1, \cdots, L$$

Since h_l^{i-1} and w_l^i are independent of each other by the Assumption 1 and the definition of h_l^{i-1}, $E[h_l^i] = n_{i-1} E[w_l^i] E[h_l^{i-1}]$. Therefore,

$$E[h_l^L] = \prod_{i=1}^{L} n_{i-1} E[w_l^i] E[\tilde{h}_l^0] E[m_l] = \prod_{i=1}^{L} n_{i-1} \mu_w^i \mu_x \mu_m$$

□

A.2 PROOF OF THEOREM 2

Proof. From the definition of $h_l, w_l^1, h_l^0, \tilde{h}_l^0, m_l$ and the property of an affine function $\sigma(E[\cdot]) = E[\sigma(\cdot)]$, the following equation holds.

$$E[h_l^1] = \sigma\left(n_0 E[w_l^1 h_l^0] + E[b_l^1]\right) = \sigma\left(n_0 E[w_l^1 \tilde{h}_l^0 m_l] + E[b_l^1]\right)$$

From the Assumption 1, w_l^1, $\tilde{h}_l^0$, and m_l are independent of each other. Thus,

$$\begin{aligned} E[h_l^1] &= \sigma\left(n_0 E[w_l^1] E[\tilde{h}_l^0] E[m_l] + E[b_l^1]\right) \\ &= \sigma\left(n_0 \mu_w^1 \mu_x \mu_m + \mu_b^1\right) \\ &= f_1(\mu_x \mu_m) \end{aligned}$$

Similarly, the following holds.

$$E[h_l^i] = \sigma\left(n_{i-1} E[w_l^i h_l^{i-1}] + E[b_l^i]\right) \text{ for } i = 1, \cdots, L$$

Since h_l^{i-1} and w_l^i are independent of each other by the Assumption 1 and the definition of h_l^{i-1},

$$\begin{aligned} E[h_l^i] &= \sigma\left(n_{i-1} E[w_l^i] E[h_l^{i-1}] + E[b_l^i]\right) \\ &= \sigma\left(n_{i-1} \mu_w^i E[h_l^{i-1}] + \mu_b^i\right) \\ &= f_i(E[h_l^i]) \end{aligned}$$

Therefore,

$$E[h_l^L] = f_L \circ \cdots \circ f_1(\mu_x \mu_m)$$

□

A.3 PROOF OF THEOREM 3

Proof. From the definition of $h_l, w_l^1, h_l^0, \tilde{h}_l^0, m_l$ and the property of a convex function $E[\sigma(\cdot)] \geq \sigma(E[\cdot])$, the following equation holds.

$$E[h_l^1] \geq \sigma\left(n_0 E[w_l^1 h_l^0] + E[b_l^1]\right) = \sigma\left(n_0 E[w_l^1 \tilde{h}_l^0 m_l] + E[b_l^1]\right)$$

From the Assumption 1, w_l^1, $\tilde{h}_l^0$, and m_l are independent of each other. Thus,

$$\begin{aligned} E[h_l^1] &\geq \sigma\left(n_0 E[w_l^1] E[\tilde{h}_l^0] E[m_l] + E[b_l^1]\right) \\ &= \sigma\left(n_0 \mu_w^1 \mu_x \mu_m + \mu_b^1\right) \\ &= f_1(\mu_x \mu_m) \end{aligned}$$

Similarly, the following holds.

$$E[h_l^i] \geq \sigma\left(n_{i-1} E[w_l^i h_l^{i-1}] + E[b_l^i]\right) \text{ for } i = 1, \cdots, L$$

Since h_l^{i-1} and w_l^i are independent of each other by thee Assumption 1 and the definition of h_l^{i-1},

$$\begin{aligned} E[h_l^i] &\geq \sigma\left(n_{i-1} E[w_l^i] E[h_l^{i-1}] + E[b_l^i]\right) \\ &= \sigma\left(n_{i-1} \mu_w^i E[h_l^{i-1}] + \mu_b^i\right) \\ &= f_i(E[h_l^i]) \end{aligned}$$

Since we assume that σ is non-decreasing, we finally get

$$E[h_l^L] \geq f_L \circ \cdots \circ f_1(\mu_x \mu_m)$$

□

A.4 PROOF OF THEOREM 4

Proof. By theorem 2, $E[h_l^L] = f_L \circ \cdots \circ f_1(E[\mathbf{h}_{\text{SN}}^0])$. Since $E[\mathbf{h}_{\text{SN}}^0] = E[\mathbf{h}^0] \cdot K/\mu_m$,

$$E[h_l^L] = f_L \circ \cdots \circ f_1(\mu_x \mu_m \cdot K/\mu_m) = f_L \circ \cdots \circ f_1(\mu_x \cdot K)$$

□

B COLLABORATIVE FILTERING (RECOMMENDATION) DATASETS

B.1 DETAILED EXPERIMENTAL SETTINGS OF TABLE 1

This subsection describes the experimental setting of a detailed collaborative filtering task in Section 4. As already mentioned, we follow the setting of AutoRec, CF-NADE as much as possible. We train neural networks with a single hidden layer with 500 units, with the weight decay regularization. For fair comparisons, we tune the hyper-parameters for weight decay in all experiments to have only one significant digit, and use a learning rate of 10^{-3} for both models[6], except for the AutoRec on Movielens 10M where we used a learning rate of 10^{-4}. We use full batch for AutoRec on Movielens 100K and 1M, mini-batch (1000) for AutoRec on Movielens 10M, and mini-batch (512) for CF-NADE models[7]. We performed five experiments, reporting mean and 95% confidence intervals. We randomly selected 10% of the ratings of each datasets for the test set (Harper & Konstan, 2016). As the dataset is too small for Movielens 100K and 1M datasets, the confidence interval tends to be large by changing the dataset split. In the results of Table 1, the same dataset split was used between 5 experiments.

Especially, for CF-NADE, they used weight sharing and averaging possible choices in addition to weight decay. We report the results of applying SN without weight sharing and averaging possible choices in Table 1. With weight sharing, it is not clear how to apply SN, so we consider the model except weight sharing. In the case of averaging possible choices, there was almost no performance gain but costs much time.

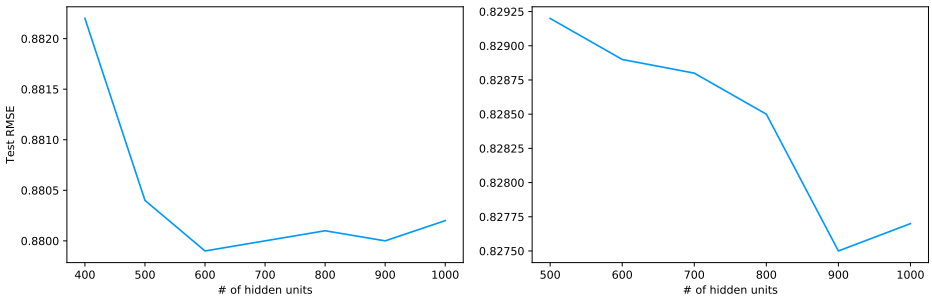


Figure 6: Test RMSE of AutoRec with SN on Movielens 100K **(left)** and Movielens 1M **(right)** with respect to the number of hidden units.

B.2 DETAILED EXPERIMENTAL SETTINGS OF TABLE 2

In comparison with other states-of-the-arts models, we used 1000 hidden units in AutoRec (Sedhain et al., 2015) with SN. While Sedhain et al. (2015) claimed that they were able to achieve enough performance only with 500 hidden units, 500 hidden units did not achieve sufficient performance when applying SN. The Figure 6 plots the test RMSE, changing the number of hidden units for Movielens 100K and 1M. We can see that 600 units for Movielens 100k and 900 units for Movielens 1M are necessary for getting better performance. Obviously, as datasets become more complex and larger, we need more network capacity. Therefore, we decide to use two times larger network capacity (1000 hidden units) to get better performance. The number of hidden units can also be viewed and tuned as a hyper-parameter, and we have not tuned much for the number of hidden units. Note that, unlike Table 1, we report the results with five random splits for all data sets in Table 2 to compare fairly with other states-of-the-arts methods.

[6]The original CF-NADE uses learning rate 5×10^{-4} for Movielens 10M, but we use 10^{-3} for fast convergence. Therefore, the results can be somewhat different from original paper.

[7]We use $K = 500$ for these experiments.

B.3 WHY WE USE ADAM OPTIMIZER IN AUTOREC?

In this subsection we explain why we use Adam Optimizer to train the AutoRec model. The authors of AutoRec used an optimizer called Resilient Propagation (RProp) to train the AutoRec model. This optimizer has a fast convergence speed but can only be used in full batch. When training Movielens 10M data using AutoRec based on item vector, training in full batch is not possible with 12GB of GPU memory. So, we decide to use Adam Optimizer. However, the performance of Adam should not be lower than that of RProp. Fortunately, we find that using Adam in most cases show similar or better performance to using RProp. The experimental results are summarized in Table 4.

Table 4: Comparison of Test RMSE between Adam and Resilient Prop Optimizer on Movielens 100K, 1M, 10M datasets.

Datasets	Movielens 100K		Movielens 1M		Movielens 10M	
input vector	item vector	user vector	item vector	user vector	item vector	user vector
RProp	0.8861	0.9437	0.8358	**0.8804**	0.782[†]	**0.867**[†]
Adam	**0.8831**	**0.9343**	**0.8306**	0.8832	**0.7807**	0.8859

[†] : Taken from Sedhain et al. (2015).

C ELECTRIC MEDICAL RECORDS (EMR) DATASETS

Table 5: Debiasing variable sparsity using SN and dropout on five disease identification tasks of NHIS dataset. Test AUROC with 95% confidence interval of 5-runs is provided.

Dataset	Cardiovascular	Fatty Liver	Hypertension	Heart Failure	Diabetes
w/o SN	0.7084 ± 0.0005	0.6858 ± 0.0065	$\mathbf{0.8023 \pm 0.0054}$	0.7876 ± 0.0012	$\mathbf{0.9263 \pm 0.0026}$
w/ SN	$\mathbf{0.7105 \pm 0.0009}$	$\mathbf{0.6941 \pm 0.0011}$	$\mathbf{0.8086 \pm 0.0016}$	$\mathbf{0.7922 \pm 0.0015}$	$\mathbf{0.9303 \pm 0.0029}$

The NHIS dataset, which is from National Health Insurance Service (NHIS), consists of medical diagnosis of around 300,000 people. The goal is to predict the occurrence of 5 diseases. Each patients takes 34 examinations over 5 years. We split dataset into two set (train and test), where the ratio of train and test split is 3:1. We preprocessed input data with min-max normalization, which makes min and max values of each features be zero and one. We train 2 layer neural networks which have 50 and 30 hidden units each, and evaluate the model with AUROC. We use ReLU activation, dropout rate 0.8 (if applied). Since these dataset is too imbalance, we apply class weight method to handle label imbalance. We use Adam optimizer with learning rate 10^{-2} and no weight decay, full batch. We evaluate the model on 5 times and report mean and 95% confidence interval. We also confirm that Sparsity Normalization makes performance gain even if we applied networks to dropout (See Table 5).

Table 6: Debiasing variable sparsity using SN on mortality prediction task of PhysioNet Challenge 2012. Test AUROC with 95% confidence interval of 5-runs is provided.

Model	Test AUROC
w/o SN	$\mathbf{0.8177 \pm 0.0071}$
w/ SN	$\mathbf{0.8152 \pm 0.0010}$

PhysioNet Challenge 2012 dataset (Silva et al., 2012) consists of 48 hours-long multivariate clinical time series data from intensive care unit (ICU). We divide 48 hours into 288 timesteps, which contain 35 examinations each. The goal of this task is to predict in-hospital death. We use dataset split given by PhysioNet Challenge 2012 (Silva et al., 2012) where each of them contains 4000 data points. In the preprocessing phase, we standardize (make mean as zero and standard deviation as one for each features) the input features and fill zero for missing values (zero imputation). We train 1 layer LSTM network which has 108 hidden units, and evaluate the model with AUROC. We apply class weight to handle imbalance problem in the dataset like in setting above. We use Adam optimizer with learning rate 2×10^{-4}, 512 batch size and early stopping method based on AUROC of validation set. We evaluate the model on 5 times and report mean and 95% confidence interval. As we mentioned in section 4.2, SN could not make significant performance gain in PhysioNet Challenge 2012 dataset (Silva et al., 2012) as shown in Table 6. Nevertheless, SN is still valuable for its ability to prevent biased predictions in this mission-critical area.

D SINGLE-CELL RNA SEQUENCE DATASETS

In the AutoImpute experiments, we run the experiments using the author's public code[8].Talwar et al. (2018) reported experimental results on 8 datasets (Blakeley, Jurkat, Kolodziejczyk, Preimplantation, Quake, Usoskin, Zeisel, and PBMC). Since we can obtain preprocessed data sets for seven data sets except PBMC, we run experiments on 7 single cell RNA sequence datasets[9]. All experimental settings without SN are exactly followed by the author's code and the hyper-parameter settings published in Table 2 of their paper. For the model applying the SN, we follow most of the experimental settings with the original model, including the hyper parameters used for hidden units or weight decay except that we use the smaller threshold with SN because the models with SN tends to be underfitted by using the threshold as suggested by the authors[10]. In addition, Talwar et al. (2018) conducted experiments only on cases with test set ratios of {0.1, 0.2, 0.3, 0.4, 0.5}, but we explored more test set ratios {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9} to show that the SN works well under more sparsity (extremely high test set ratio). Like the authors' settings, we perform 10 experiments and report mean and 95% confidence intervals. We change the test set split on every trials. It is remarkable that although hyper-parameters are not favorable for SN, SN performed similarly or better on all datasets.

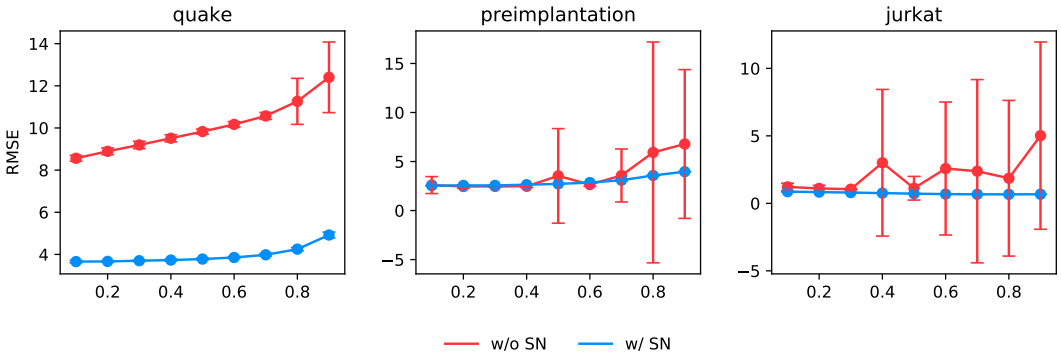


Figure 7: Debiasing variable sparsity using SN according to test set ratio on three imputation tasks of single cell RNA sequence dataset. Test RMSE with 95% confidence interval of 10-runs is provided.

E DROPOUT ON UCI DATASETS

As we metioned on Section 4.4, most experimental settings are adopted from Klambauer et al. (2017); Littwin & Wolf (2018)'s UCI experiments. We used ReLU (Glorot et al., 2011) networks of 4 hidden layers with 256 units. We use Mean Square Error (MSE) for loss function and Adam optimizer with no weight decay, $\epsilon = 10^{-8}$ and learning rate 10^{-4}. The batch size used for training is 128. We split 10% from the dataset for test and use 20% of training set as the validation set. For all inputs, we applied min-max normalization, which makes min 0 and max 1. Note that, we use dense datasets with no missing inputs to focus on the effects of dropout. All the datasets are used as provided in the package in `sklearn.datasets`.

[8] https://github.com/divyanshu-talwar/AutoImpute

[9] https://drive.google.com/drive/folders/1q2ho_cNfsQJNbdCt9j0nwlZv-Roj_yK1

[10] One more thing to note that we use $\lambda = 20$ for Preimplantation dataset with SN

F DENSITY ESTIMATION

To reproduce binarized MNIST experiments of MADE (Germain et al., 2015), we have adopted and fixed a public implementation of MADE[11]. The figure 5 shows our reproduction results of Figure 2 in original paper. We follow most settings of the MADE. We use a single hidden layer MADE networks with 500 units, learning rate 0.005, and test on authors' binarized MNIST dataset[12]. The only difference from the original authors' implementation is that we used Adam, rather using Adagrad. When using Adagrade, we found that the learning rate dropped so rapidly that learning did not work. In our implementation, we replaced the optimizer with Adam ($\epsilon = 10^{-8}$, and no weight decay).

Since there is no missingness in the Binarized MNIST data set, it cannot be divided by $\|M\|_1$ as suggested by Algorithm 1. In this experiment, we regard $\|M\|_1$ as $\|\mathbf{h}^0\|_0$ so as not to lose the generality. That is, all pixels that are 0 in the binarized MNIST dataset are regarded as missing. We label results of this with `w/SN` and plot them in the figure 5. We mention in the section 4.5 that MADE can also cause variable sparsity by weight. In MADE, the connection between specific units is forcibly controlled through a mechanism of element-wise product of a specific mask matrix M^i in weight $W^i \in \mathbb{R}^{n_i \times n_{i-1}}$. These mask matrices also cause variation in sparsity. We plot the results of using the new mask matrix M^i_{SN} in place of the mask matrix M^i used by MADE with `w/SN(all)` in figure 5. The method of calculating the new mask matrix M^i_{SN} using the existing mask matrix M^i is as follows:

$$M^i_{\text{SN}} \leftarrow (\mathbf{1}^T M^i \mathbf{1}/n_i) \cdot M^i \oslash (M^i \mathbf{1}\mathbf{1}^T)$$

where $\oslash$ denotes element-wise division and $\mathbf{1}$ is a column vector where all elements are 1. $(\mathbf{1}^T M^i \mathbf{1}/n_i)$ corresponds to K' and $(M^i \mathbf{1}\mathbf{1}^T)$ does to $\|\mathbf{m}\|_1$ in Algorithm 1.

We have demonstrated the effectiveness of SN even in situations where the learning rate is smaller. Smaller learning rates shows the effect of SN more clearly despite of their longer training time. We show the effect of SN on MADE when the learning rate is 0.001 in Figure 8.

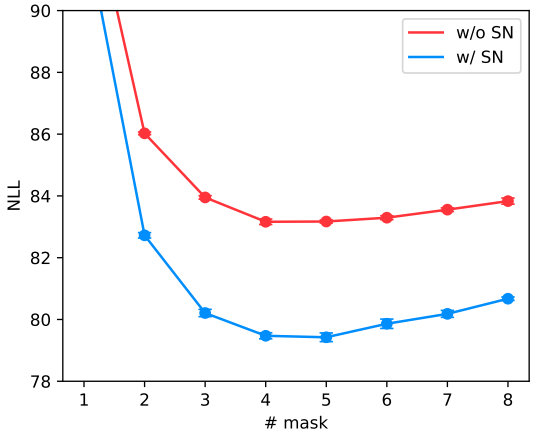


Figure 8: Negative loglikelihood of MADE on binarized MNIST with and without SN. We use Adam optimizer with learning rate 0.001 rather Adagrad.

G MACHINE DESCRIPTION

We perform all the experiments on a Titan X with 12GB of VRAM. 12 GB of VRAM is not always necessary, and most experiments require smaller VRAM.

[11] `https://github.com/karpathy/pytorch-made`

[12] `https://github.com/mgermain/MADE/releases/download/ICML2015/binarized_mnist.npz`