Title
End-to-End Machine Learning Frameworks for Medicine: Data Imputation, Model Interpretation and Synthetic Data Generation

Author
Yoon, Jinsung

Publication Date
2020

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
Los Angeles

End-to-End Machine Learning Frameworks for Medicine: Data Imputation, Model Interpretation and Synthetic Data Generation

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Electrical and Computer Engineering

by

Jinsung Yoon

2020

ABSTRACT OF THE DISSERTATION

End-to-End Machine Learning Frameworks for Medicine:
Data Imputation, Model Interpretation and Synthetic Data Generation

by

Jinsung Yoon
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2020
Professor Mihaela van der Schaar, Chair

Tremendous successes in machine learning have been achieved in a variety of applications such as image classification and language translation via supervised learning frameworks. Recently, with the rapid increase of electronic health records (EHR), machine learning researchers got immense opportunities to adopt the successful supervised learning frameworks to diverse clinical applications. To properly employ machine learning frameworks for medicine, we need to handle the special properties of the EHR and clinical applications: (1) extensive missing data, (2) model interpretation, (3) privacy of the data. This dissertation addresses those specialties to construct end-to-end machine learning frameworks for clinical decision support.

We focus on the following three problems: (1) how to deal with incomplete data (data imputation), (2) how to explain the decisions of the trained model (model interpretation), (3) how to generate synthetic data for better sharing private clinical data (synthetic data generation). To appropriately handle those problems, we propose novel machine learning algorithms for both static and longitudinal settings. For data imputation, we propose modified Generative Adversarial Networks and Recurrent Neural Networks to accurately impute the missing values and return the complete data for applying state-of-the-art supervised learning models. For model interpretation, we utilize the actor-critic framework to estimate feature importance of the trained model's decision in an instance level. We expand this algorithm to active sensing framework that recommends which observations should we measure and

when. For synthetic data generation, we extend well-known Generative Adversarial Network frameworks from static setting to longitudinal setting, and propose a novel differentially private synthetic data generation framework.

To demonstrate the utilities of the proposed models, we evaluate those models on various real-world medical datasets including cohorts in the intensive care units, wards, and primary care hospitals. We show that the proposed algorithms consistently outperform state-of-the-art for handling missing data, understanding the trained model, and generating private synthetic data that are critical for building end-to-end machine learning frameworks for medicine.

The dissertation of Jinsung Yoon is approved.

Gregory J. Pottie

Jonathan Chau-Yan Kao

William Hsu

Mihaela van der Schaar, Committee Chair

University of California, Los Angeles

2020

To my parents, my brother's family, and my love

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

It would not have been possible to complete this doctoral dissertation without the help and support of the kind people around me. I am deeply grateful.

First, I would like to express the deepest appreciation to my advisor, Professor Mihaela van der Schaar, for her expertise, assistance, and patience throughout my entire PhD research. Her unwavering enthusiasm for research kept me constantly engaged with my research on developing diverse machine learning algorithms for medicine. Without her devoted guidance, persistent help, and insightful research directions, this dissertation would not have been possible. I would also thank my dissertation committee members, Professor Gregory Pottie, Professor Jonathan Kao, and Professor William Hsu for their thoughtful suggestions and considerate comments.

My sincere thanks also goes to all of my colleagues. I have been lucky to spend some time in Europe as a recognized student at University of Oxford; thus, my colleagues are spread around the globe. I would like to thank all of my co-authors; Professor William Zame, Professor Cem Tekin, Dr. Ahmed Alaa, Changhee Lee, and Kyeong Ho Kenneth Moon at UCLA; James Jordon at University of Oxford; Daniel Jarrett, Yao Zhang, and Dr. Lydia Drumright at University of Cambridge. It has been a great honor to collaborate with such brilliant people. I would also like to thank all of my lab mates; Dr. William Whoiles, Dr. Kartik Ahuja, and Trent Kyono at UCLA; Ioana Bica at University of Oxford; Alexis Bellot, Alihan Huyuk, and Zhaozhi Qian at University of Cambridge for their critical comments and passionate research discussions.

I was fortunate to collaborate with many clinicians who tremendously help transforming my research into real-world clinical applications. I would like to thank Professor Raffaele Bugiardini (University of Bologna, Italy) for providing precious clinical datasets across the entire Europe, guiding to come up with the vital clinical discovery, and publishing high-impact clinical journals such as JAMA Internal Medicine. I would also like to thank other clinical collaborators; Dr. Scott Hu, Dr. Camelia Davtyan, Dr. Martin Cadeiras, and Dr. Mindy Ross at UCLA; Dr. Lydia Drumright and Dr. Ari Ercole at University of Cambridge for

their sympathetic clinical comments and suggestions on my research.

Finally, and most importantly, I wish to thank my parents, my brother's family, and my love, for their endless support. I have no valuable words to express my thanks for their spiritual supports throughout my entire graduate education.

VITA

2014	Bachelor in Electrical Engineering and Computer Engineering Department, Seoul National University, Seoul, Korea
2016	Master of Science in Electrical and Computer Engineering Department, University of California, Los Angeles, United States
2015–2020	Graduate Student Researcher in Electrical and Computer Engineering Department, University of California, Los Angeles, United States.
2015–2016	Teaching Assistant, Electrical and Computer Engineering Department, University of California, Los Angeles, United States.
2016	Outstanding Master Thesis, Electrical and Computer Engineering Department, University of California, Los Angeles, United States.
2017–2018	Recognized Student, Engineering Science Department, University of Oxford, Oxfordshire, United Kingdom.
2019	Research Intern, Google Cloud AI, California, United States.

PUBLICATIONS

J. Yoon, L. N. Drumright, M. van der Schaar, " Anonymization through Data Synthesis using Generative Adversarial Networks (ADS-GAN)," *IEEE J. Biomedical and Health Informatics* (JBHI), 2020.

J. Yoon, D. Jarrett, M. van der Schaar, "Time-series Generative Adversarial Networks," *Neural Information Processing Systems* (NeurIPS), 2019.

J. Yoon, J. Jordon, M. van der Schaar, "INVASE: Instance-wise Variable Selection using Neural Networks," *International Conference on Learning Representations* (ICLR), 2019.

J. Yoon, J. Jordon, M. van der Schaar, "PATE-GAN: Generating Synthetic Data with Differential Privacy Guarantees," *International Conference on Learning Representations* (ICLR), 2019.

J. Yoon, J. Jordon, M. van der Schaar, "GAIN: Missing Data Imputation using Generative Adversarial Nets," *International Conference on Machine Learning* (ICML), 2018.

J. Yoon, J. Jordon, M. van der Schaar, "RadialGAN: Leveraging Multiple Datasets to Improve Target-specific Predictive Models using Generative Adversarial Networks," *International Conference on Machine Learning* (ICML), 2018.

J. Yoon, J. Jordon, M. van der Schaar, "GANITE: Estimation of Individualized Treatment Effects using Generative Adversarial nets," *International Conference on Learning Representations* (ICLR), 2018.

J. Yoon, W. R. Zame, M. van der Schaar, "Deep Sensing: Active Sensing using Multi-directional Recurrent Neural Networks," *International Conference on Learning Representations* (ICLR), 2018.

J. Yoon, W. R. Zame, A. Banerjee, M. Cadeiras, A. M. Alaa, M. van der Schaar, "Personalized Survival Predictions via Trees of Predictors: An Application to Cardiac Transplantation," *PloS One*, 2018.

J. Yoon, W. R. Zame, M. van der Schaar, "Estimating Missing Data in Temporal Data Streams using Multi-directional Recurrent Neural Networks," *IEEE Transactions on Biomedical Engineering* (TBME), 2018.

J. Yoon, W. R. Zame, M. van der Schaar, "ToPs: Ensemble Learning with Trees of Predictors," *IEEE Transactions on Signal Processing* (TSP), 2018.

J. Yoon, A. M. Alaa, M. Cadeiras, M. van der Schaar, "Personalized Donor-Recipient Matching for Organ Transplantation," *AAAI*, 2017.

J. Yoon, A. M. Alaa, S. Hu, M. van der Schaar, "ForecastICU: A Prognostic Decision Support System for Timely Prediction of Intensive Care Unit Admission," *International Conference on Machine Learning* (ICML), 2016.

J. Yoon, C. Davtyan, M. van der Schaar, "Discovery and Clinical Decision Support for Personalized Healthcare," *IEEE Journal of Biomedical and Health Informatics* (JBHI), 2015.

CHAPTER 1

Introduction

With the advent of electronic health records (EHR), the collections of clinical data are rapidly increased for numerous patients across the entire world. Simultaneously, data-driven machine learning frameworks have been achieved enormous successes in a variety of applications (such as image classification [HZR16], object detection [HGD17], and language translation [VSP17]) with deep learning models via supervised learning frameworks.

The availability of various medical datasets and high performing machine learning frameworks results in extensive opportunities for developing diverse data-driven clinical decision support such as early warning systems [YAH16] and clinical risk scoring systems [AS18a]. However, unlike image and language domains, medical domain has its own characteristics that machine learning researchers must consider to constructing end-to-end machine learning frameworks for medicine.

First, missing data is ubiquitous in medical data. The missing data problem is especially challenging in medical domains which present time-series containing many streams of measurements that are sampled at different and irregular times [YZS18a]. This is significantly important because accurate estimation of these missing measurements is often critical for accurate diagnosis, prognosis [AS18b] and treatment, as well as for accurate modeling and statistical analyses [AYH18].

Second, clinical decision support should provide proper interpretations for its decisions. Medicine is more conservative field than computer vision and natural language. Without proper understanding of the trained models and explanations of their decisions, it is difficult to widely apply those models to real-world clinical decision support. Clinicians expect to get not only patient outcome predictions but also how those predictions are derived from clinical

decision support.

Third, medical data is usually private. Medical and machine learning communities are relying on the promise of artificial intelligence (AI) to transform medicine through enabling more accurate decisions and personalized treatment. However, progress is slow. Legal and ethical issues around unconsented patient data and privacy is one of the limiting factors in data sharing, resulting in a significant barrier in accessing routinely collected EHR by the machine learning community. To alleviate this difficulty, generating synthetic data that closely approximates the joint distribution of variables in an original EHR dataset can provide a readily accessible, legally and ethically appropriate solution to support more open data sharing, and enable the development of AI solutions.

In this Chapter, we illustrate the end-to-end machine learning pipeline for healthcare application, and synthetic data generation framework for private medical data sharing. Then, we summarize the contributions of the following chapters in this dissertation.

1.1 End-to-end machine learning pipeline for medicine

High-level end-to-end machine learning pipeline is made up of three stages: (1) Data preprocessing, (2) Model training, (3) Model interpretation. Most machine learning researchers focus on stage (2) - model training via supervised learning frameworks. For instance, popular image classification models such as ResNet [HZR16] and InceptionV3 [SVI16] are convolutional neural networks based supervised learning model for stage (2). On the other hand, stage (1) and stage (3) are under-explored even though those stages are critical in medicine. In this dissertation, we focus on developing novel and high-performing machine learning algorithms for stage (1) and (3). Fig. 1.1 illustrate the high-level abstractions of the end-to-end machine learning pipeline in longitudinal setting.

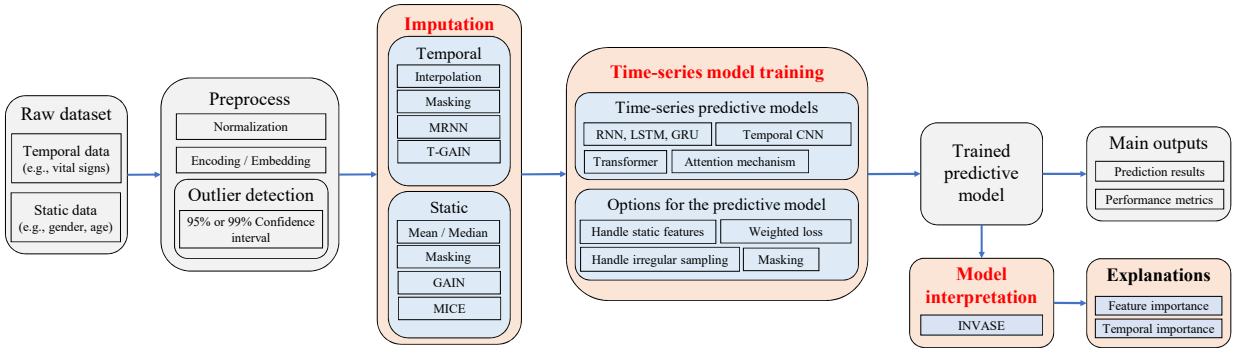

Figure 1.1: End-to-end machine learning pipeline in longitudinal setting. (1) Data preprocessing (including imputation), (2) Model training, (3) Model interpretation.

1.1.1 Data imputation

Missing data is a pervasive problem. Data may be missing for many reasons. For instance, in the medical domain, the respiratory rate of a patient may not have been measured (perhaps because it was deemed unnecessary/unimportant) or accidentally not recorded [YDS17, AYH18]. It may also be the case that certain pieces of information are difficult or even dangerous to acquire (such as information gathered from a biopsy), and so these were not gathered for those reasons [YZB18]. The critical part of the medical data preprocessing stage is how to deal with those missing values. Accurate estimation of the missing measurements is important for many reasons, including diagnosis, prognosis and treatment. Furthermore, most state-of-the-art machine learning models are only applicable when the input data is complete; thus, without proper handling of the missing data, we cannot move on to the next stage (training machine learning models) of the machine learning pipeline.

In Chapter 2 and 3, we propose state-of-the-art imputation models for static and longitudinal settings. In Chapter 2, we propose a novel imputation method using modified Generative Adversarial Networks in static setting. In Chapter 3, we modified Recurrent Neural Networks for imputing missing data in longitudinal setting.

1.1.2 Model interpretation

State-of-the-art prediction performance is not the only expectation for clinical decision support. Reasonable explanations of the decisions are mandatory for doctors and patients to trust the clinical decision support. Moreover, understanding data-driven machine learning models for medicine can provide new insights on medicine which may result in the vital clinical discovery.

Defining the model interpretability is not straightforward, and there are various definitions of model interpretability such as symbolic modeling [AS19] and concept-based modeling [KWG17]. In this dissertation, we define the model interpretability as discovering instance-wise feature importance which is widely used interpretation definition [SGK17, CSW18].

In Chapter 4, we proposed an instance-wise feature selection method for interpreting the trained model using a novel actor-critic framework. This can provide an explanation (i.e. evidence or support) of the trained model's individual decision.

In Chapter 5, we extend the instance-wise feature selection method to active sensing problem. In many medical settings, making observations is costly [WRG96]. For example, performing lab tests on a patient incurs a cost, both financially as well as causing fatigue to the patient [KBR09, KNS08]. In such settings, the decision to observe is important and should be an active choice. We propose a novel actor-critic model of recommending which measurements should we measure and when.

1.2 Synthetic data generation for private data sharing

The adoption of EHR has dramatically increased in high-income countries over the last decade [HPS17, KJP17, GH16] with corresponding interest to do so in low and middle income countries worldwide [LB15]. Evidence from both small scale studies and other disciplines suggests that machine learning could support significant advances in healthcare delivery [RFP19, TP18], however, appropriate legal and ethical management of routinely collected EHR can create obstacles to open sharing of sensitive health data.

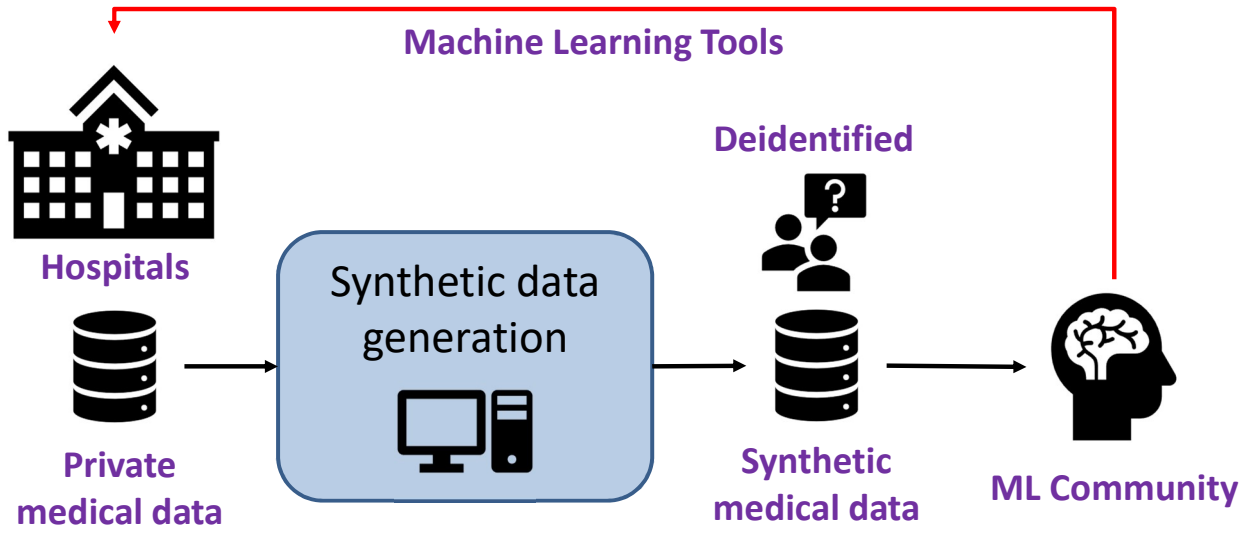

Figure 1.2: Synthetic data generation for sharing the private medical data to machine learning community for developing machine learning tools easier.

Given the complicated dynamics around protecting, anonymizing and sharing routinely collected health data, we decided to address the problem in a new way. We developed a model to create entirely synthetic datasets of individuals that are fictious and yet could be drawn from the same population as the real dataset. Fig. 1.2 illustrates the simple block diagram of synthetic data generation for private data sharing between machine learning community and clinical data providers.

In Chapter 6, we extends well-known Generative Adversarial Networks for synthetic data generation from static setting to longitudinal setting. In Chapter 7, we proposed a differentially private synthetic data generation framework where differential privacy is well-defined mathematical notion of the privacy [DR14].

1.3 Summary of contributions

In this section, we summarize the contributions of the following chapters in this dissertation.

1.3.1 Chapter 2 contributions

In Chapter 2, we consider the missing data imputation problem in static setting. We propose a novel method for imputing missing data by adapting the well-known Generative Adversarial Nets (GAN) framework. The generator observes some components of a real data vector, imputes the missing components conditioned on what is actually observed, and outputs a completed vector. The discriminator then takes a completed vector and attempts to determine which components were actually observed and which were imputed. To ensure that the discriminator forces the generator to learn the desired distribution, we provide the discriminator with some additional information in the form of a hint vector. The hint reveals to the discriminator partial information about the missingness of the original sample, which is used by the discriminator to focus its attention on the imputation quality of particular components. This hint ensures that the generator does in fact learn to generate according to the true data distribution.

1.3.2 Chapter 3 contributions

In Chapter 3, we address the missing data imputation problem in longitudinal setting. Existing methods address this estimation problem by interpolating within data streams or imputing across data streams (both of which ignore important information) or ignoring the temporal aspect of the data and imposing strong assumptions about the nature of the data-generating process and/or the pattern of missing data (both of which are especially problematic for medical data). We propose a new approach, based on a novel deep learning architecture that interpolates within data streams and imputes across data streams.

1.3.3 Chapter 4 contributions

In Chapter 4, we tackle the model interpretation problem in static setting where we define the interpretation as estimating instance-wise feature importance for individual prediction. We propose a new instance-wise feature selection method. The proposed model consists of 3 neural networks, a selector network, a predictor network and a baseline network which are

used to train the selector network using the actor-critic framework. Using this methodology, the proposed model is capable of flexibly discovering feature subsets of a different size for each instance, which is a key limitation of existing state-of-the-art methods.

1.3.4 Chapter 5 contributions

In Chapter 5, we extend the instance-wise feature selection methodology to active sensing problem in the longitudinal setting. Deciding what and when to observe is critical when making observations is costly. In a medical setting where observations can be made sequentially, making these observations (or not) should be an active choice. We propose a novel deep learning framework to address this problem. The proposed model consists of two networks: a selector network and a predictor network. The selector network uses previously selected observations to determine what should be observed in the future. The predictor network uses the observations selected by the selector network to predict a label, providing feedback to the selector network (well-selected variables should be predictive of the label). The goal of the selector network is then to select variables that balance the cost of observing the selected variables with their predictive power.

1.3.5 Chapter 6 contributions

In Chapter 6, we study the synthetic data generation problem in longitudinal setting. A good generative model for time-series data should preserve temporal dynamics, in the sense that new sequences respect the original relationships between variables across time. Existing methods that bring GANs into the sequential setting do not adequately attend to the temporal correlations unique to time-series data. At the same time, supervised models for sequence prediction - which allow finer control over network dynamics - are inherently deterministic. We propose a novel framework for generating realistic time-series data that combines the flexibility of the unsupervised paradigm with the control afforded by supervised training. Through a learned embedding space jointly optimized with both supervised and adversarial objectives, we encourage the network to adhere to the dynamics of the training data during

sampling.

1.3.6 Chapter 7 contributions

In Chapter 7, we focus on the private synthetic data generation problem in static setting. We investigate a method for ensuring differential privacy of the generator of the GAN framework. The resulting model can be used for generating synthetic data on which algorithms can be trained and validated, and on which competitions can be conducted, without compromising the privacy of the original dataset. Our method modifies the Private Aggregation of Teacher Ensembles framework and applies it to GANs. We also look at measuring the quality of synthetic data from a new angle; we assert that for the synthetic data to be useful for machine learning researchers, the relative performance of two algorithms (trained and tested) on the synthetic dataset should be the same as their relative performance (when trained and tested) on the original dataset.

CHAPTER 2

GAIN: Missing Data Imputation using Generative Adversarial Nets

Missing values are prevalent in medical data. Data may be missing because it was never collected, records were lost or for many other reasons. An imputation algorithm can be used to estimate missing values based on data that was observed/measured, such as the systolic blood pressure and heart rate of the patient [YZS18a]. A substantial amount of research has been dedicated to developing imputation algorithms for medical data [BM99, Mac10, SWC09, PS15]. Imputation algorithms are also used in many other applications such as image concealment, data compression, and counterfactual estimation [Rub04, KL12, YJS18].

State-of-the-art imputation methods can be categorized as either discriminative or generative. Discriminative methods include MICE [WRW11, BG11], MissForest [SB11], and matrix completion [CR09, YRD16, SSS16]; generative methods include algorithms based on Expectation Maximization [GSF10] and algorithms based on deep learning (e.g. denoising autoencoders (DAE) and generative adversarial nets (GAN)) [VLB08, GW18, AL16]. However, current generative methods for imputation have various drawbacks. For instance, the approach for data imputation based on [GSF10] makes assumptions about the underlying distribution and fails to generalize well when datasets contain mixed categorical and continuous variables. In contrast, the approaches based on DAE [VLB08] have been shown to work well in practice but require complete data during training. In many circumstances, missing values are part of the inherent structure of the problem so obtaining a complete dataset is impossible. Another approach with DAE [GW18] allows for an incomplete dataset; however, it only utilizes the observed components to learn the representations of the data. [AL16] uses Deep Convolutional GANs for image completion; however, it also requires complete data for

training the discriminator.

In this chapter, we propose a novel imputation method, which we call Generative Adversarial Imputation Nets (GAIN), that generalizes the well-known GAN [GPM14] and is able to operate successfully even when complete data is unavailable. In GAIN, the generator's goal is to accurately impute missing data, and the discriminator's goal is to distinguish between observed and imputed components. The discriminator is trained to minimize the classification loss (when classifying which components were observed and which have been imputed), and the generator is trained to maximize the discriminator's misclassification rate. Thus, these two networks are trained using an adversarial process. To achieve this goal, GAIN builds on and adapts the standard GAN architecture. To ensure that the result of this adversarial process is the desired target, the GAIN architecture provides the discriminator with additional information in the form of "hints". This hinting ensures that the generator generates samples according to the true underlying data distribution.

2.1 Background: Three types of missing data - MCAR, MAR, and MNAR

Missing data can be categorized into three types: (1) the data is missing completely at random (MCAR) if the missingness occurs entirely at random (there is no dependency on any of the variables), (2) the data is missing at random (MAR) if the missingness depends only on the *observed* variables, (3) the data is missing not at random (MNAR) if the missingness is neither MCAR nor MAR (more specifically, the data is MNAR if the missingness depends on *both observed* variables and the *unobserved* variables; thus, missingness cannot be fully accounted for by the observed variables). A formal definition of MCAR, MAR, and MNAR can be found in the subsequent subsection. In this chapter we provide theoretical results for our algorithm under the MCAR assumption, and empirically compare to other state-of-the-art methods in all three settings (MCAR, MAR, and MNAR). Here we recall the definition of the first two, and formalize the other.

MCAR: Data is said to be Missing Completely at Random (MCAR) if:

$$\mathbf{X} \perp\!\!\!\perp \mathbf{M} \tag{2.1}$$

MAR: Data is said to be Missing at Random (MAR) if:

$$\begin{aligned} &\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}, \mathbf{m} \in \{0,1\}^d \text{ s.t. } \tilde{\mathbf{x}}_1 = \tilde{\mathbf{x}}_2 \text{ (w.r.t. } \mathbf{m}) \\ &\mathbb{P}(\mathbf{M} = \mathbf{m} | \mathbf{X} = \mathbf{x}_1) = \mathbb{P}(\mathbf{M} = \mathbf{m} | \mathbf{X} = \mathbf{x}_2) \end{aligned} \tag{2.2}$$

MNAR: Data is said to be Missing Not at Random (MNAR) if it is neither MCAR or MAR (in particular, the missingness can depend on the values of the *unobserved* data points).

2.2 Problem formulation

Consider a d-dimensional space $\mathcal{X} = \mathcal{X}_1 \times ... \times \mathcal{X}_d$. Suppose that $\mathbf{X} = (X_1, ..., X_d)$ is a random variable (either continuous or binary) taking values in $\mathcal{X}$, whose distribution we will denote $P(\mathbf{X})$. Suppose that $\mathbf{M} = (M_1, ..., M_d)$ is a random variable taking values in $\{0,1\}^d$. We will call $\mathbf{X}$ the data vector, and $\mathbf{M}$ the mask vector.

For each $i \in \{1, ..., d\}$, we define a new space $\tilde{\mathcal{X}}_i = \mathcal{X}_i \cup \{*\}$ where $*$ is simply a point not in any $\mathcal{X}_i$, representing an unobserved value. Let $\tilde{\mathcal{X}} = \tilde{\mathcal{X}}_1 \times ... \times \tilde{\mathcal{X}}_d$. We define a new random variable $\tilde{\mathbf{X}} = (\tilde{X}_1, ..., \tilde{X}_d) \in \tilde{\mathcal{X}}$ in the following way:

$$\tilde{X}_i = \begin{cases} X_i, & \text{if } M_i = 1 \\ *, & \text{otherwise} \end{cases} \tag{2.3}$$

so that $\mathbf{M}$ indicates which components of $\mathbf{X}$ are observed. Note that we can recover $\mathbf{M}$ from $\tilde{\mathbf{X}}$.

Throughout the remainder of the chapter, we will often use lower-case letters to denote realizations of a random variable and use the notation $\mathbf{1}$ to denote a vector of 1s, whose dimension will be clear from the context (most often, d).

2.2.1 Imputation

In the imputation setting, n i.i.d. copies of $\tilde{\mathbf{X}}$ are realized, denoted $\tilde{\mathbf{x}}^1, ..., \tilde{\mathbf{x}}^n$ and we define the dataset $\mathcal{D} = \{(\tilde{\mathbf{x}}^i, \mathbf{m}^i)\}_{i=1}^n$, where $\mathbf{m}^i$ is simply the recovered realization of $\mathbf{M}$ corresponding to $\tilde{\mathbf{x}}^i$.

Our goal is to *impute* the unobserved values in each $\tilde{\mathbf{x}}_i$. Formally, we want to generate samples according to $P(\mathbf{X}|\tilde{\mathbf{X}} = \tilde{\mathbf{x}}^i)$, the conditional distribution of $\mathbf{X}$ given $\tilde{\mathbf{X}} = \tilde{\mathbf{x}}^i$, for each i, to fill in the missing data points in $\mathcal{D}$. By attempting to model the *distribution* of the data rather than just the expectation, we are able to make multiple draws and therefore make *multiple imputations* allowing us to capture the uncertainty of the imputed values [WRW11, BG11, Rub04].

2.3 GAIN: Generative Adversarial Imputation Nets

In this section we describe our approach for simulating $P(\mathbf{X}|\tilde{\mathbf{X}} = \tilde{\mathbf{x}}^i)$ which is motivated by GANs. We highlight key similarities and differences to a standard (conditional) GAN throughout. Fig. 2.1 depicts the overall architecture of GAIN.

2.3.1 Generator

The generator, G, takes (realizations of) $\tilde{\mathbf{X}}$, $\mathbf{M}$ and a noise variable, $\mathbf{Z}$, as input and outputs $\bar{\mathbf{X}}$, a vector of imputed values. Let $G : \tilde{\mathcal{X}} \times \{0,1\}^d \times [0,1]^d \to \mathcal{X}$ be a function, and $\mathbf{Z} = (Z_1, ..., Z_d)$ be d-dimensional noise (independent of all other variables).

Then we define the random variables $\bar{\mathbf{X}}, \hat{\mathbf{X}} \in \mathcal{X}$ by

$$\bar{\mathbf{X}} = G(\tilde{\mathbf{X}}, \mathbf{M}, (\mathbf{1} - \mathbf{M}) \odot \mathbf{Z}) \tag{2.4}$$

$$\hat{\mathbf{X}} = \mathbf{M} \odot \tilde{\mathbf{X}} + (\mathbf{1} - \mathbf{M}) \odot \bar{\mathbf{X}} \tag{2.5}$$

where $\odot$ denotes element-wise multiplication. $\bar{\mathbf{X}}$ corresponds to the vector of *imputed* values (note that G outputs a value for every component, even if its value was observed) and $\hat{\mathbf{X}}$

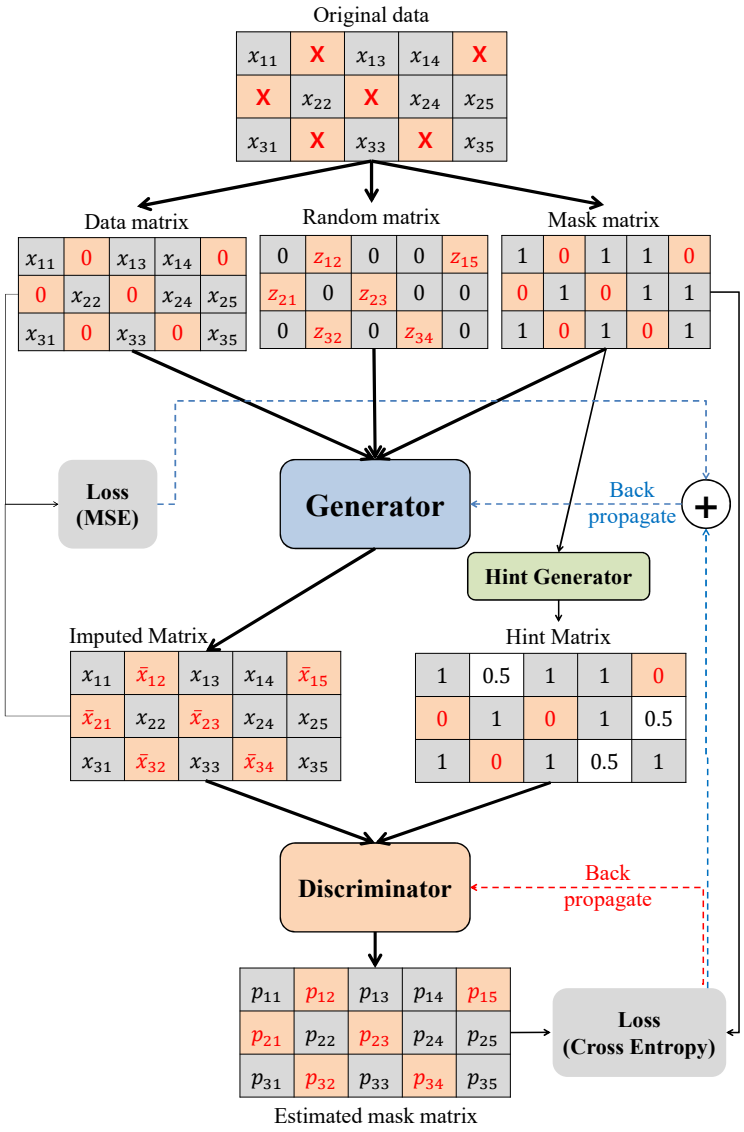

Figure 2.1: The architecture of GAIN with exemplar samples.

corresponds to the completed data vector, that is, the vector obtained by taking the partial observation $\tilde{\mathbf{X}}$ and replacing each $*$ with the corresponding value of $\bar{\mathbf{X}}$.

This setup is very similar to a standard GAN, with $\mathbf{Z}$ being analogous to the noise variables introduced in that framework. Note, though, that in this framework, the target distribution, $P(\mathbf{X}|\tilde{\mathbf{X}})$, is essentially $||\mathbf{1} - \mathbf{M}||_1$-dimensional and so the noise we pass into the generator is $(\mathbf{1} - \mathbf{M}) \odot \mathbf{Z}$, rather than simply $\mathbf{Z}$, so that its dimension matches that of the targeted distribution.

2.3.2 Discriminator

As in the GAN framework, we introduce a discriminator, D, that will be used as an adversary to train G. However, unlike in a standard GAN where the output of the generator is either *completely* real or *completely* fake, in this setting the output is comprised of some components that are real and some that are fake. Rather than identifying that an entire vector is real or fake, the discriminator attempts to distinguish which *components* are real (observed) or fake (imputed) - this amounts to predicting the mask vector, $\mathbf{m}$. Note that the mask vector $\mathbf{M}$ is pre-determined by the dataset.

Formally, the discriminator is a function $D : \mathcal{X} \to [0,1]^d$ with the i-th component of $D(\hat{\mathbf{x}})$ corresponding to the probability that the i-th component of $\hat{\mathbf{x}}$ was observed.

2.3.3 Hint

As will be seen in the theoretical results that follow, it is necessary to introduce what we call a hint mechanism. A hint mechanism is a random variable, $\mathbf{H}$, taking values in a space $\mathcal{H}$, both of which *we define*. We allow $\mathbf{H}$ to depend on $\mathbf{M}$ and for each (imputed) sample $(\hat{\mathbf{x}}, \mathbf{m})$, we draw $\mathbf{h}$ according to the distribution $\mathbf{H}|\mathbf{M} = \mathbf{m}$. We pass $\mathbf{h}$ as an additional input to the discriminator and so it becomes a function $D : \mathcal{X} \times \mathcal{H} \to [0,1]^d$, where now the i-th component of $D(\hat{\mathbf{x}}, \mathbf{h})$ corresponds to the probability that the i-th component of $\hat{\mathbf{x}}$ was observed conditional on $\hat{\mathbf{X}} = \hat{\mathbf{x}}$ *and* $\mathbf{H} = \mathbf{h}$.

By defining $\mathbf{H}$ in different ways, we control the amount of information contained in $\mathbf{H}$ about $\mathbf{M}$ and in particular we show (in Proposition 1) that if we do not provide "enough" information about $\mathbf{M}$ to D (such as if we simply did not have a hinting mechanism), then there are several distributions that G could reproduce that would all be optimal with respect to D.

2.3.4 Objective

We train D to *maximize* the probability of correctly predicting $\mathbf{M}$. We train G to *minimize* the probability of D predicting $\mathbf{M}$. We define the quantity $V(D, G)$ to be

$$V(D,G) = \mathbb{E}_{\hat{\mathbf{X}},\mathbf{M},\mathbf{H}}\Big[\mathbf{M}^T \log D(\hat{\mathbf{X}}, \mathbf{H}) + (\mathbf{1} - \mathbf{M})^T \log\big(\mathbf{1} - D(\hat{\mathbf{X}}, \mathbf{H})\big)\Big], \tag{2.6}$$

where log is element-wise logarithm and dependence on G is through $\hat{\mathbf{X}}$.

Then, as with the standard GAN, we define the objective of GAIN to be the minimax problem given by

$$\min_G \max_D V(D,G). \tag{2.7}$$

We define the loss function $\mathcal{L} : \{0,1\}^d \times [0,1]^d \to \mathbb{R}$ by

$$\mathcal{L}(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{d} \Big[a_i \log(b_i) + (1 - a_i) \log(1 - b_i)\Big]. \tag{2.8}$$

Writing $\hat{\mathbf{M}} = D(\hat{\mathbf{X}}, \mathbf{H})$, we can then rewrite (2.7) as

$$\min_G \max_D \mathbb{E}\big[\mathcal{L}(\mathbf{M}, \hat{\mathbf{M}})\big]. \tag{2.9}$$

2.4 Theoretical analysis

In this section we provide a theoretical analysis of Equation (2.7). Given a d-dimensional space $\mathcal{Z} = \mathcal{Z}_1 \times ... \times \mathcal{Z}_d$, a (probability) density[1] p over $\mathcal{Z}$ corresponding to a random variable Z, and a vector $\mathbf{b} \in \{0,1\}^d$ we define the set $A_\mathbf{b} = \{i : b_i = 1\}$, the projection $\phi_\mathbf{b} : \mathcal{Z} \to \Pi_{i \in A_\mathbf{b}} \mathcal{Z}_i$ by $\phi_\mathbf{b}(z) = (z_i)_{i \in A}$ and the density $p^\mathbf{b}$ to be the density of $\phi_\mathbf{b}(Z)$.

Throughout this section, we make the assumption that $\mathbf{M}$ is independent of $\mathbf{X}$, i.e. that the data is MCAR.

We will write $p(\mathbf{x}, \mathbf{m}, \mathbf{h})$ to denote the density of the random variable $(\hat{\mathbf{X}}, \mathbf{M}, \mathbf{H})$ and we

[1]For ease of exposition, we use the term density even when referring to a probability mass function.

will write $\hat{p}$, p_m and p_h to denote the marginal densities (of p) corresponding to $\hat{\mathbf{X}}$, $\mathbf{M}$ and $\mathbf{H}$, respectively. When referring to the joint density of two of the three variables (potentially conditioned on the third), we will simply use p, abusing notation slightly.

It is more intuitive to think of this density through its decomposition into densities corresponding to the true data generating process, and to the generator defined by Equation (2.4),

$$p(\mathbf{x}, \mathbf{m}, \mathbf{h}) = p_m(\mathbf{m})\hat{p}^{\mathbf{m}}(\phi_{\mathbf{m}}(\mathbf{x}|\mathbf{m})) \times \hat{p}^{\mathbf{1}-\mathbf{m}}(\phi_{\mathbf{1}-\mathbf{m}}(\mathbf{x})|\mathbf{m}, \phi_{\mathbf{m}}(\mathbf{x}))p_h(\mathbf{h}|\mathbf{m}). \tag{2.10}$$

The first two terms in Equation (2.10) are both defined by the data, where $\hat{p}^{\mathbf{m}}(\phi_{\mathbf{m}}(\mathbf{x})|\mathbf{m})$ is the density of $\phi_{\mathbf{m}}(\hat{\mathbf{X}})|\mathbf{M} = \mathbf{m}$ which corresponds to the density of $\phi_{\mathbf{m}}(\mathbf{X})$ (i.e. the true data distribution), since conditional on $\mathbf{M} = \mathbf{m}$, $\phi_{\mathbf{m}}(\hat{\mathbf{X}}) = \phi_{\mathbf{m}}(\mathbf{X})$ (see Equations (2.3) and (2.5)). The third term, $\hat{p}^{\mathbf{1}-\mathbf{m}}(\phi_{\mathbf{1}-\mathbf{m}}(\mathbf{x})|\mathbf{m}, \phi_{\mathbf{m}}(\mathbf{x}))$, is determined by the generator, G, and is the density of the random variable $\phi_{\mathbf{1}-\mathbf{m}}(G(\tilde{\mathbf{x}}, \mathbf{m}, \mathbf{Z})) = \phi_{\mathbf{1}-\mathbf{m}}(\bar{\mathbf{X}})|\tilde{\mathbf{X}} = \tilde{\mathbf{x}}, \mathbf{M} = \mathbf{m}$ where $\tilde{\mathbf{x}}$ is determined by $\mathbf{m}$ and $\phi_{\mathbf{m}}(\mathbf{x})$. The final term is the conditional density of the hint, which we are free to define (its selection will be motivated by the following analysis).

Using this decomposition, one can think of drawing a sample from $\hat{p}$ as first sampling $\mathbf{m}$ according to $p_m(\cdot)$, then sampling the "observed" components, $\mathbf{x}_{obs}$, according to $\hat{p}^{\mathbf{m}}(\cdot)$ (we can then construct $\tilde{\mathbf{x}}$ from $\mathbf{x}_{obs}$ and $\mathbf{m}$), then *generating* the imputed values, $\mathbf{x}_{imp}$, from the generator according to $\hat{p}^{\mathbf{1}-\mathbf{m}}(\cdot|\mathbf{m}, \mathbf{x}_{obs})$ and finally sampling the hint according to $p_h(\cdot|\mathbf{m})$.

Lemma 1. *Let $\mathbf{x} \in \mathcal{X}$. Let p_h be a fixed density over the hint space $\mathcal{H}$ and let $\mathbf{h} \in \mathcal{H}$ be such that $p(\mathbf{x}, \mathbf{h}) > 0$. Then for a fixed generator, G, the i-th component of the optimal discriminator, $D^*(\mathbf{x}, \mathbf{h})$ is given by*

$$D^*(\mathbf{x}, \mathbf{h})_i = \frac{p(\mathbf{x}, \mathbf{h}, m_i = 1)}{p(\mathbf{x}, \mathbf{h}, m_i = 1) + p(\mathbf{x}, \mathbf{h}, m_i = 0)} = p_m(m_i = 1|\mathbf{x}, \mathbf{h}) \tag{2.11}$$

for each $i \in \{1, ..., d\}$.

We now rewrite Equation (2.6), substituting for D^*, to obtain the following minimization

criterion for G:

$$C(G) = \mathbb{E}_{\hat{\mathbf{X}},\mathbf{M},\mathbf{H}}\Big(\sum_{i:M_i=1} \log p_m(m_i = 1|\hat{\mathbf{X}}, \mathbf{H}) + \sum_{i:M_i=0} \log p_m(m_i = 0|\hat{\mathbf{X}}, \mathbf{H})\Big), \tag{2.12}$$

where dependence on G is through $p_m(\cdot|\hat{\mathbf{X}})$.

Theorem 1. *A global minimum for $C(G)$ is achieved if and only if the density $\hat{p}$ satisfies*

$$\hat{p}(\mathbf{x}|\mathbf{h}, m_i = t) = \hat{p}(\mathbf{x}|\mathbf{h}) \tag{2.13}$$

for each $i \in \{1, ..., d\}$, $\mathbf{x} \in \mathcal{X}$ and $\mathbf{h} \in \mathcal{H}$ such that $p_h(\mathbf{h}|m_i = t) > 0$.

The following proposition asserts that if $\mathbf{H}$ does not contain "enough" information about $\mathbf{M}$, we cannot guarantee that G learns the desired distribution (the one uniquely defined by the (underlying) data).

Proposition 1. *There exist distributions of $\mathbf{X}$, $\mathbf{M}$ and $\mathbf{H}$ for which solutions to Equation (2.13) are not unique. In fact, if $\mathbf{H}$ is independent of $\mathbf{M}$, then Equation (2.13) does not define a unique density, in general.*

Let the random variable $\mathbf{B} = (B_1, ..., B_d) \in \{0, 1\}^d$ be defined by first sampling k from $\{1, ..., d\}$ uniformly at random and then setting

$$B_j = \begin{cases} 1 \text{ if } j \neq k \\ 0 \text{ if } j = k. \end{cases} \tag{2.14}$$

Let $\mathcal{H} = \{0, 0.5, 1\}^d$ and, given $\mathbf{M}$, define

$$\mathbf{H} = \mathbf{B} \odot \mathbf{M} + 0.5(\mathbf{1} - \mathbf{B}). \tag{2.15}$$

Observe first that $\mathbf{H}$ is such that $H_i = t \implies M_i = t$ for $t \in \{0, 1\}$ but that $H_i = 0.5$ implies nothing about M_i. In other words, $\mathbf{H}$ reveals all but one of the components of $\mathbf{M}$ to D. Note,

however, that $\mathbf{H}$ does contain some information about M_i since M_i is not assumed to be independent of the other components of $\mathbf{M}$.

The following lemma confirms that the discriminator behaves as we expect with respect to this hint mechanism.

Lemma 2. *Suppose* $\mathbf{H}$ *is defined as above. Then for* $\mathbf{h}$ *such that* $h_i = 0$ *we have* $D^*(\mathbf{x}, \mathbf{h})_i = 0$ *and for* $\mathbf{h}$ *such that* $h_i = 1$ *we have* $D^*(\mathbf{x}, \mathbf{h})_i = 1$*, for all* $\mathbf{x} \in \mathcal{X}$*,* $i \in \{1, ..., d\}$*.*

The final proposition we state tells us that $\mathbf{H}$ as specified above ensures the generator learns to replicate the desired distribution.

Proposition 2. *Suppose* $\mathbf{H}$ *is defined as above. Then the solution to Equation (2.13) is unique and satisfies*

$$\hat{p}(\mathbf{x}|\mathbf{m}_1) = \hat{p}(\mathbf{x}|\mathbf{m}_2) \tag{2.16}$$

for all $\mathbf{m}_1, \mathbf{m}_2 \in \{0, 1\}^d$*. In particular,* $\hat{p}(\mathbf{x}|\mathbf{m}) = \hat{p}(\mathbf{x}|\mathbf{1})$ *and since* $\mathbf{M}$ *is independent of* $\mathbf{X}$*,* $\hat{p}(\mathbf{x}|\mathbf{1})$ *is the density of* $\mathbf{X}$*. The distribution of* $\hat{\mathbf{X}}$ *is therefore the same as the distribution of* $\mathbf{X}$*.*

For the remainder of the chapter, $\mathbf{B}$ and $\mathbf{H}$ will be defined as in Equations (2.14) and (2.15).

2.5 GAIN algorithm

Using an approach similar to that in [GPM14], we solve the minimax optimization problem (Equation (2.7)) in an iterative manner. Both G and D are modeled as fully connected neural nets.

We first optimize the discriminator D with a fixed generator G using mini-batches of size k_D. For each sample in the mini-batch[2], $(\tilde{\mathbf{x}}(j), \mathbf{m}(j))$, we draw k_D independent samples, $\mathbf{z}(j)$ and $\mathbf{b}(j)$, of $\mathbf{Z}$ and $\mathbf{B}$ and compute $\hat{\mathbf{x}}(j)$ and $\mathbf{h}(j)$ accordingly. Lemma 2 then tells us that

[2]The index j now corresponds to the j-th sample of the mini-batch, rather than the j-th sample of the entire dataset.

the only outputs of D that depend on G are the ones corresponding to $b_i = 0$ for each sample. We therefore only train D to give us these outputs (if we also trained D to match the outputs specified in Lemma 2 we would gain no information about G, but D would overfit to the hint vector). We define $\mathcal{L}_D : \{0,1\}^d \times [0,1]^d \times \{0,1\}^d \to \mathbb{R}$ by

$$\mathcal{L}_D(\mathbf{m}, \hat{\mathbf{m}}, \mathbf{b}) = \sum_{i:b_i=0} \Big[m_i \log(\hat{m}_i) + (1 - m_i) \log(1 - \hat{m}_i) \Big]. \tag{2.17}$$

D is then trained according to

$$\min_D - \sum_{j=1}^{k_D} \mathcal{L}_D(\mathbf{m}(j), \hat{\mathbf{m}}(j), \mathbf{b}(j)) \tag{2.18}$$

recalling that $\hat{\mathbf{m}}(j) = D(\hat{\mathbf{x}}(j), \mathbf{m}(j))$.

Second, we optimize the generator G using the newly updated discriminator D with mini-batches of size k_G. We first note that G in fact outputs a value for the *entire* data vector (including values for the components we observed). Therefore, in training G, we not only ensure that the imputed values for missing components ($m_j = 0$) successfully fool the discriminator (as defined by the minimax game), we also ensure that the values outputted by G for observed components ($m_j = 1$) are close to those actually observed. This is justified by noting that the conditional distribution of $\mathbf{X}$ given $\tilde{\mathbf{X}} = \tilde{\mathbf{x}}$ obviously fixes the components of $\mathbf{X}$ corresponding to $M_i = 1$ to be $\tilde{X}_i$. This also ensures that the representations learned in the hidden layers of $\tilde{\mathbf{X}}$ suitably capture the information contained in $\tilde{\mathbf{X}}$ (as in an auto-encoder).

To achieve this, we define two different loss functions. The first, $\mathcal{L}_G : \{0,1\}^d \times [0,1]^d \times \{0,1\}^d \to \mathbb{R}$, is given by

$$\mathcal{L}_G(\mathbf{m}, \hat{\mathbf{m}}, \mathbf{b}) = - \sum_{i:b_i=0} (1 - m_i) \log(\hat{m}_i), \tag{2.19}$$

and the second, $\mathcal{L}_M : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, by

$$\mathcal{L}_M(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{d} m_i L_M(x_i, x_i'), \tag{2.20}$$

where

$$L_M(x_i, x_i') = \begin{cases} (x_i' - x_i)^2, & \text{if } x_i \text{ is continuous,} \\ -x_i \log(x_i'), & \text{if } x_i \text{ is binary.} \end{cases}$$

As can be seen from their definitions, $\mathcal{L}_G$ will apply to the missing components ($m_i = 0$) and $\mathcal{L}_M$ will apply to the observed components ($m_i = 1$).

$\mathcal{L}_G(\mathbf{m}, \hat{\mathbf{m}})$ is smaller when $\hat{m}_i$ is closer to 1 for i such that $m_i = 0$. That is, $\mathcal{L}_G(\mathbf{m}, \hat{\mathbf{m}})$ is smaller when D is less able to identify the imputed values as being imputed (it falsely categorizes them as observed). $\mathcal{L}_M(\mathbf{x}, \tilde{\mathbf{x}})$ is minimized when the reconstructed features (i.e. the values G outputs for features that were observed) are close to the actually observed features.

G is then trained to minimize the weighted sum of the two losses as follows:

$$\min_G \sum_{j=1}^{k_G} \mathcal{L}_G(\mathbf{m}(j), \hat{\mathbf{m}}(j), \mathbf{b}(j)) + \alpha \mathcal{L}_M(\tilde{\mathbf{x}}(j), \hat{\mathbf{x}}(j)),$$

where α is a hyper-parameter.

The pseudo-code of GAIN is presented in Algorithm 1.

2.6 Experiments

In this section, we validate the performance of GAIN using multiple real-world datasets. In the first set of experiments we qualitatively analyze the properties of GAIN. In the second we quantitatively evaluate the imputation performance of GAIN using various UCI datasets [Lic13], giving comparisons with state-of-the-art imputation methods. In the third we evaluate the performance of GAIN in various settings (such as on datasets with different missing

Algorithm 1 Pseudo-code of GAIN

while training loss has not converged **do**

(1) Discriminator optimization

Draw k_D samples from the dataset $\{(\tilde{\mathbf{x}}(j), \mathbf{m}(j))\}_{j=1}^{k_D}$

Draw k_D i.i.d. samples, $\{\mathbf{z}(j)\}_{j=1}^{k_D}$, of $\mathbf{Z}$

Draw k_D i.i.d. samples, $\{\mathbf{b}(j)\}_{j=1}^{k_D}$, of $\mathbf{B}$

for $j = 1, ..., k_D$ **do**

$\bar{\mathbf{x}}(j) \leftarrow G(\tilde{\mathbf{x}}(j), \mathbf{m}(j), \mathbf{z}(j))$

$\hat{\mathbf{x}}(j) \leftarrow \mathbf{m}(j) \odot \tilde{\mathbf{x}}(j) + (\mathbf{1} - \mathbf{m}(j)) \odot \bar{\mathbf{x}}(j)$

$\mathbf{h}(j) = \mathbf{b}(j) \odot \mathbf{m}(j) + 0.5(\mathbf{1} - \mathbf{b}(j))$

end for

Update D using stochastic gradient descent (SGD)

$$\nabla_D - \sum_{j=1}^{k_D} \mathcal{L}_D(\mathbf{m}(j), D(\hat{\mathbf{x}}(j), \mathbf{h}(j)), \mathbf{b}(j))$$

(2) Generator optimization

Draw k_G samples from the dataset $\{(\tilde{\mathbf{x}}(j), \mathbf{m}(j))\}_{j=1}^{k_G}$

Draw k_G i.i.d. samples, $\{\mathbf{z}(j)\}_{j=1}^{k_G}$ of $\mathbf{Z}$

Draw k_G i.i.d. samples, $\{\mathbf{b}(j)\}_{j=1}$ of $\mathbf{B}$

for $j = 1, ..., k_G$ **do**

$\mathbf{h}(j) = \mathbf{b}(j) \odot \mathbf{m}(j) + 0.5(\mathbf{1} - \mathbf{b}(j))$

end for

Update G using SGD (for fixed D)

$$\nabla_G \sum_{j=1}^{k_G} \mathcal{L}_G(\mathbf{m}(j), \hat{\mathbf{m}}(j), \mathbf{b}(j)) + \alpha \mathcal{L}_M(\mathbf{x}(j), \tilde{\mathbf{x}}(j))$$

end while=0

rates). In the final set of experiments we evaluate GAIN against other imputation algorithms when the goal is to perform prediction on the imputed dataset.

We conduct each experiment 10 times and within each experiment we use 5-cross validations. We report either RMSE or AUROC as the performance metric along with their standard deviations across the 10 experiments. Unless otherwise stated, missingness is applied to the datasets by randomly removing 20% of all data points (MCAR).

In all experiments, the depth of the generator and discriminator in both GAIN and auto-encoder is set to 3. The number of hidden nodes in each layer is d, $d/2$ and d,

Algorithm	Breast	Spam	Credit	News
GAIN	**.0546 ± .0006**	**.0513± .0016**	**.1858 ± .0010**	**.1441 ± .0007**
GAIN w/o $\mathcal{L}_G$	.0701 ± .0021 **(22.1%)**	.0676 ± .0029 **(24.1%)**	.2436 ± .0012 **(23.7%)**	.1612 ± .0024 **(10.6%)**
GAIN w/o $\mathcal{L}_M$	.0767 ± .0015 **(28.9%)**	.0672 ± .0036 **(23.7%)**	.2533 ± .0048 **(26.7%)**	.2522 ± .0042 **(42.9%)**
GAIN w/o Hint	.0639 ± .0018 **(14.6%)**	.0582 ± .0008 **(11.9%)**	.2173 ± .0052 **(14.5%)**	.1521 ± .0008 **(5.3%)**
GAIN w/o Hint & $\mathcal{L}_M$	.0782 ± .0016 **(30.1%)**	.0700 ± .0064 **(26.7%)**	.2789 ± .0071 **(33.4%)**	.2527 ± .0052 **(43.0%)**

Table 2.1: Source of gains in GAIN algorithm (Mean ± Std of RMSE (Gain (%)))

respectively. We use tanh as the activation functions of each layer except for the output layer where we use the sigmoid activation function and the number of batches is 64 for both the generator and discriminator. For the GAIN algorithm, we use cross-validation to select α among $\{0.1, 0.5, 1, 2, 10\}$. Implementation of GAIN can be found at `https://github.com/jsyoon0823/GAIN`.

2.6.1 Source of gain

The potential sources of gain for the GAIN framework are: the use of a GAN-like architecture (through $\mathcal{L}_G$), the use of reconstruction error in the loss ($\mathcal{L}_M$), and the use of the hint ($\mathbf{H}$). In order to understand how each of these affects the performance of GAIN, we exclude one or two of them and compare the performances of the resulting architectures against the full GAIN architecture.

Table 2.1 shows that the performance of GAIN is improved when all three components are included. More specifically, the full GAIN framework has a 15% improvement over the simple auto-encoder model (i.e. GAIN w/o $\mathcal{L}_G$). Furthermore, utilizing the hint vector additionally gives improvements of 10%.

2.6.2 Quantitative analysis of GAIN

We use four real-world datasets from UCI Machine Learning Repository [Lic13] (Breast, Spam, Credit, and News) to quantitatively evaluate the imputation performance of GAIN. Details of each dataset are reported in Table 2.2.

Dataset	**N**	$\mathbf{S_{cont}}$	$\mathbf{S_{cat}}$	**Average Correlations**
Breast	569	30	0	0.3949
Spam	4,601	57	0	0.0608
Credit	30,000	14	9	0.1633
News	39,797	44	14	0.0688

Table 2.2: Statistics of the datasets. S_{cont}: the number of continuous variables, S_{cat}: the number of categorical variables

In Table 2.3 we report the RMSE (and its standard deviation) for GAIN and 5 other state-of-the-art imputation methods: MICE [WRW11, BG11], MissForest [SB11], Matrix completion (Matrix) [CR09], Auto-encoder [GW18] and Expectation-maximization (EM) [GSF10]. As can be seen from the table, GAIN significantly outperforms each benchmark across all 4 datasets

2.6.3 GAIN in different settings

To better understand GAIN, we conduct several experiments in which we vary the missing rate, the number of samples, and the number of dimensions using Credit dataset. Fig. 2.2 shows the performance (RMSE) of GAIN within these different settings in comparison to the two most competitive benchmarks (MissForest and Auto-encoder). Fig. 2.2 (a) shows that, even though the performance of each algorithm decreases as missing rates increase, GAIN consistently outperforms the benchmarks across the entire range of missing rates.

Fig. 2.2 (b) shows that as the number of samples increases, the performance improvements of GAIN over the benchmarks also increases. This is due to the large number of parameters in GAIN that need to be optimized, however, as demonstrated on the Breast dataset (in Table 2.3), GAIN is still able to outperform the benchmarks even when the number of samples is

Algorithm	Breast	Spam	Credit	News
GAIN	**.0546 ± .0006**	**.0513± .0016**	**.1858 ± .0010**	**.1441 ± .0007**
MICE	.0646 ± .0028	.0699 ± .0010	.2585 ± .0011	.1763 ± .0007
MissForest	.0608 ± .0013	.0553 ± .0013	.1976 ± .0015	.1623 ± 0.012
Matrix	.0946 ± .0020	.0542 ± .0006	.2602 ± .0073	.2282 ± .0005
Auto-encoder	.0697 ± .0018	.0670 ± .0030	.2388 ± .0005	.1667 ± .0014
EM	.0634 ± .0021	.0712 ± .0012	.2604 ± .0015	.1912 ± .0011

Table 2.3: Imputation performance in terms of RMSE (Average ± Std of RMSE)

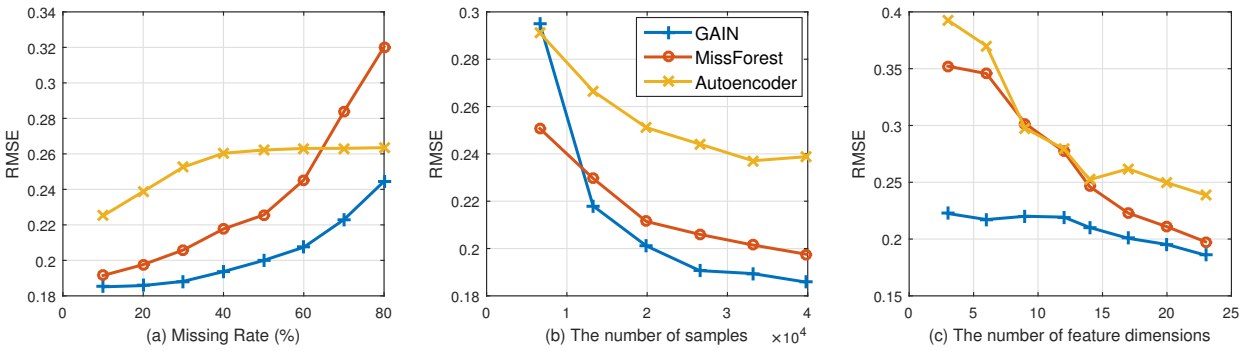

Figure 2.2: RMSE performance in different settings: (a) Various missing rates, (b) Various number of samples, (c) Various feature dimensions

relatively small (less than 600).

Fig. 2.2 (c) shows that GAIN is also robust to the number of feature dimensions. On the other hand, the discriminative model (MissForest) cannot as easily cope when the number of feature dimensions is small.

2.6.4 GAIN in MAR and MNAR settings

In the previous subsections, we only evaluate GAIN on MCAR setting. In this subsection, we demonstrate the outperformance of GAIN on MAR and MNAR settings as well. The following explains how we constructed datasets that satisfy MAR and MNAR settings.

Missing at random (MAR): To create an MAR dataset, we sequentially define the probability that the ith component of the nth sample is observed conditional on the missingness

and values (if observed) of the previous $i-1$ components to be

$$P_i^m(n) = \frac{p^m(i) \cdot N \cdot e^{-\sum_{j<i} w_j m_j(n) x_j(n) + b_j(1-m_j(n))}}{\sum_{l=1}^{N} e^{-\sum_{j<i} w_j m_j(l) x_j(l) + b_j(1-m_j(l))}}$$

where $p^m(i)$ corresponds to the average missing rate of the ith feature, and w_j, b_j are sampled from $\mathcal{U}(0,1)$ (but are only sampled once for the entire dataset). We sequentially sample $m_1, ..., m_d$ for each feature vector.

Missing not at random (MNAR): To create an MNAR dataset, we define the probability that the ith component of the nth sample is observed ($P_i^m(n)$) to be

$$P_i^m(n) = \frac{p^m(i) \cdot N \cdot e^{-w_i x_i(n)}}{\sum_{l=1}^{N} e^{-w_i x_i(l)}}$$

where again $p^m(i)$ corresponds to the average missing rate of the ith feature and w_i is sampled from $\mathcal{U}(0,1)$. In particular, the missingness of a data point is directly dependent on its value (with dependence determined by the weight w_i).

We compare the RMSE of GAIN against other imputation algorithms on both an MAR and MNAR version of the Credit dataset. To make a fair comparison, we pass the mask matrix to all the benchmarks as an additional input so that they can also utilize the informative missingness captured by it.

Different missing rates for different features: In order to also explore the effect of different missing rates across features on the imputation performance of GAIN, we compare the MCAR, MAR and MNAR settings when $p^m(i) = 0.2 \; \forall i \in \{1, ..., d\}$ (uniform) and when $p^m(i) = 0.4 \times \hat{p}^m(i)$ where $\hat{p}^m(i) \sim \mathcal{U}(0,1)$ (non-uniform). The average missing rate in both cases is 0.2.

As can be seen in Table 2.4, GAIN outperforms other state-of-the-art imputation methods in all three missingness settings (both when feature missingness is uniform and non-uniform) and shows significantly better performance in the MNAR setting.

As can also be seen from the bottom side of Table 2.4, GAIN still outperforms all benchmarks in the non-uniform setting, although the performance of both GAIN and MissForest

Setting	Uniform		
	MCAR	**MAR**	**MNAR**
GAIN	**.1858 ± .0010**	**.1974 ± .0006**	**.4046 ± .0053**
MICE	.2585 ± .0011	.2574 ± .0035	.5310 ± .0207
MissForest	.1976 ± .0015	.2194 ± .0065	.4286 ± .0087
Matrix	.2602 ± .0073	.2473 ± .0070	.4328 ± .0036
Auto-encoder	.2388 ± .0005	.2405 ± .0070	.4876 ± .0097
EM	.2604 ± .0015	.2755 ± .0063	.5157 ± .0039
Setting	**Non-uniform**		
	MCAR	**MAR**	**MNAR**
GAIN	**.2114 ± .0007**	**.2245 ± .0008**	**.4672 ± .0066**
MICE	.2574 ± .0014	.2344 ± .0068	.5355 ± .0036
MissForest	.2496 ± .0065	.2537 ± .0097	.4784 ± .0102
Matrix	.2356 ± .0022	.2440 ± .0122	.5216 ± .0084
Auto-encoder	.2444 ± .0037	.2498 ± .0129	.5017 ± .0078
EM	.2620 ± .0010	.3339 ± .0024	.4998 ± .0053

Table 2.4: Imputation performance with uniform and non-uniform $p^m(i)$ on MCAR, MAR, and MNAR (Average ± Std of RMSE) settings.

(its closest competitor in the uniform setting) both decrease similarly, while MICE and Matrix completion both show improvements for the non-uniform setting.

Note that the standard deviation of the total number of missing points is higher for non-uniform $p^m(i)$ than uniform $p^m(i)$. As consistent with Fig. 2.2 (a), higher/lower missing rates yield higher/lower imputation errors; and so, due to the increased standard deviation, there is a greater variance in the performance in the non-uniform setting.

2.6.5 Prediction performance

We now compare GAIN against the same benchmarks with respect to the accuracy of post-imputation prediction. For this purpose, we use Area Under the Receiver Operating

Characteristic Curve (AUROC) as the measure of performance. To be fair to all methods, we use the same predictive model (logistic regression) in all cases. Comparisons are made on all datasets and the results are reported in Table 2.5.

Algorithm	**AUROC (Average ± Std)**			
	Breast	**Spam**	**Credit**	**News**
GAIN	**.9930 ± .0073**	**.9529 ± .0023**	**.7527 ± .0031**	**.9711 ± .0027**
MICE	.9914 ± .0034	.9495 ± .0031	.7427 ± .0026	.9451 ± .0037
MissForest	.9860 ± .0112	.9520 ± .0061	.7498 ± .0047	.9597 ± .0043
Matrix	.9897 ± .0042	.8639 ± .0055	.7059 ± .0150	.8578 ± .0125
Auto-encoder	.9916 ± .0059	.9403 ± .0051	.7485 ± .0031	.9321 ± .0058
EM	.9899 ± .0147	.9217 ± .0093	.7390 ± .0079	.8987 ± .0157

Table 2.5: Prediction performance comparison

As Table 2.5 shows, GAIN, which we have already shown to achieve the best imputation accuracy (in Table 2.3), yields the best post-imputation prediction accuracy. However, even in cases where the improvement in imputation accuracy is large, the improvements in prediction accuracy are not always significant. This is probably due to the fact that there is sufficient information in the (80%) observed data to predict the label.

Prediction accuracy with various missing rates: In this experiment, we evaluate the post-imputation prediction performance when the missing rate of the dataset is varied. Note that every dataset has their own binary label.

The results of this experiment (for GAIN and the two most competitive benchmarks) are shown in Fig. 2.3. In particular, the performance of GAIN is significantly better than the other two for higher missing rates, this is due to the fact that as the information contained in the observed data decreases (due to more values being missing), the imputation quality becomes more important, and GAIN has already been shown to provide (significantly) better quality imputations.

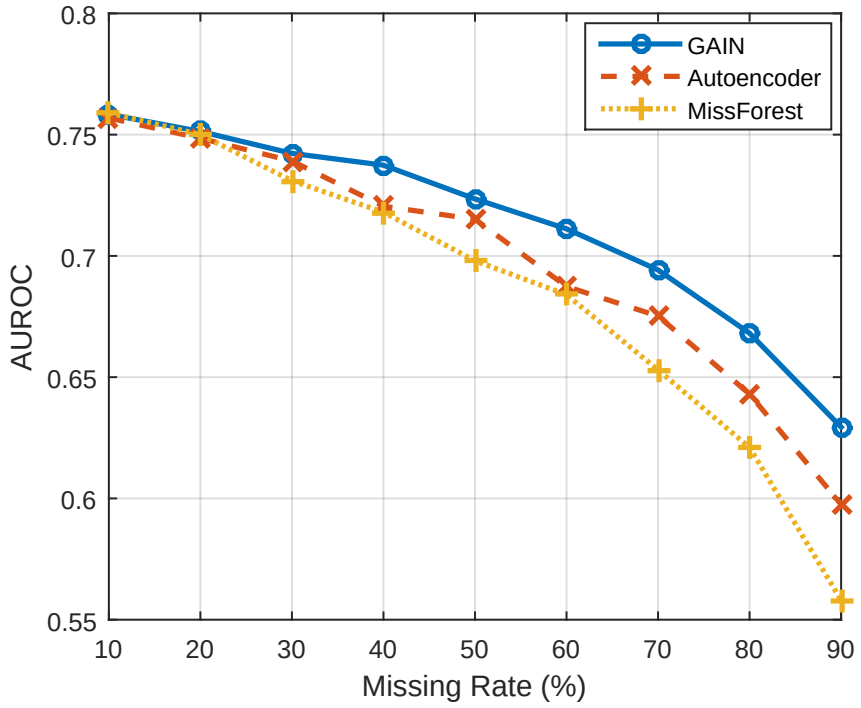

Figure 2.3: The AUROC performance with various missing rates with Credit dataset

2.6.6 Congeniality of GAIN

The congeniality of an imputation model is its ability to impute values that respect the feature-label relationship [Men94, BWR13, DCI16]. The congeniality of an imputation model can be evaluated by measuring the effects on the feature-label relationships after the imputation. We compare the logistic regression parameters, $\mathbf{w}$, learned from the complete Credit dataset with the parameters, $\hat{\mathbf{w}}$, learned from an incomplete Credit dataset by first imputing and then performing logistic regression.

We report the mean and standard deviation of both the mean bias ($||\mathbf{w} - \hat{\mathbf{w}}||_1$) and the mean square error ($||\mathbf{w} - \hat{\mathbf{w}}||_2$) for each method in Table 2.6. These quantities being lower indicates that the imputation algorithm better respects the relationship between feature and label. As can be seen in the table, GAIN achieves significantly lower mean bias and mean square error than other state-of-the-art imputation algorithms (from 8.9% to 79.2% performance improvements).

Algorithm	**Mean Bias** ($||\mathbf{w} - \hat{\mathbf{w}}||_1$)	**MSE** ($||\mathbf{w} - \hat{\mathbf{w}}||_2$)
GAIN	**0.3163± 0.0887**	**0.5078± 0.1137**
MICE	0.8315 ± 0.2293	0.9467 ± 0.2083
MissForest	0.6730 ± 0.1937	0.7081 ± 0.1625
Matrix	1.5321 ± 0.0017	1.6660 ± 0.0015
Auto-encoder	0.3500 ± 0.1503	0.5608 ±0.1697
EM	0.8418 ± 0.2675	0.9369 ± 0.2296

Table 2.6: Congeniality performances of imputation models

2.7 Conclusion

In this chapter, we propose a generative model for missing data imputation, GAIN. This novel architecture generalizes the well-known GAN such that it can deal with the unique characteristics of the imputation problem. Various experiments with real-world datasets show that GAIN significantly outperforms state-of-the-art imputation techniques. The development of a new, state-of-the-art technique for imputation can have transformative impacts; most datasets in medicine as well as in other domains have missing data.

CHAPTER 3

Estimating Missing Data in Temporal Data Streams Using Multi-directional Recurrent Neural Networks

Missing data/measurements present a ubiquitous problem. The problem is especially challenging in medical settings which present time series containing many streams of measurements that are sampled at different and irregular times [YZS18a], and is especially important in these settings because accurate estimation of these missing measurements is often critical for accurate diagnosis, prognosis [AS18b] and treatment, as well as for accurate modeling and statistical analyses [YAH16]. This chapter presents a new method for estimating missing measurements in time series data, based on a novel deep learning architecture. By comparing our method with current state-of-the-art benchmarks on a variety of real-world medical datasets, we demonstrate that our method is much more accurate in estimating missing measurements, and that this accuracy is reflected in improved prediction of outcomes.

The most familiar methods for estimating missing data follow one of three approaches, usually called *interpolation, imputation* and *matrix completion*. Interpolation methods such as [KL12, MP10] exploit the correlation among measurements at different times *within each stream* but ignore the correlation across streams. Imputation methods such as [Rub04, GSF10, WRW11, SB11] exploit the correlation among measurements at the same time *across different streams* but ignore the correlation within streams. Because medical measurements are frequently correlated both within streams *and* across streams (e.g., blood pressure at a given time is correlated both with blood pressure at other times and with heart rate), each of these approaches loses potentially important information. Matrix completion methods such as [CR09, YRD16, SSS16] do exploit correlations within and across streams, but assume that the data is static – hence ignore the temporal component of the data – or that the

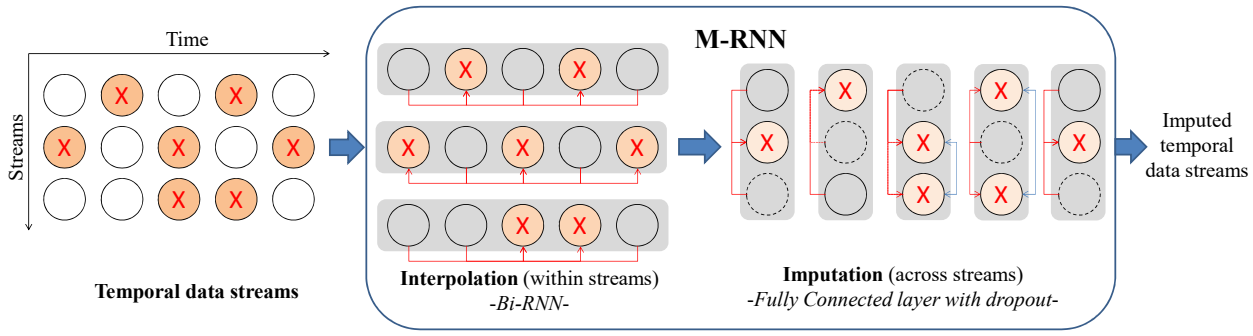

Figure 3.1: Block diagram of missing data estimation process. X: missing measurements; red lines: connections between observed values and missing values in each layer; blue lines: connections between interpolated values; dashed lines: dropout

data is perfectly synchronized – an assumption that is routinely violated in medical time series data. Some of these methods also make modeling assumptions about the nature of the data-generating process or of the pattern of missing data. Our approach is expressly designed to exploit both the correlation within streams and the correlation across streams and to take into account the temporal and non-synchronous character of the data; our approach makes no modeling assumptions about the data-generating process or the pattern of missing data. (We do assume – as is standard in most of the literature – that the data is *missing at random* [KL12]. Dealing with data that is not missing at random [AHS17] presents additional challenges.)

Our method relies on a novel neural network architecture that we call a *Multi-directional Recurrent Neural Network (M-RNN)*. Our M-RNN contains both an interpolation block and an imputation block and it trains these blocks *simultaneously*, rather than separately (See Fig. 3.1 and 3.2). Like a bi-directional RNN (Bi-RNN) [GS05], an M-RNN operates forward and backward *within* each data stream – in the *intra-stream directions*. An M-RNN also operates *across* different data streams – in the *inter-stream directions*. Unlike a Bi-RNN, the timing of inputs into the hidden layers of our M-RNN is lagged in the forward direction and advanced in the backward direction. As illustrated in Fig. 3.2, our M-RNN architecture exploits the 3-dimensional nature of the dataset.

An important aspect of medical data is that there is often enormous uncertainty in the measured data. As is well-known, although single imputation (SI) methods may yield the

most plausible/most likely estimate for each missing data point [DHS06], they do not capture the uncertainty in the imputed data [Rub04]. Multiple imputation (MI) methods capture this uncertainty by sampling imputed values several times in order to form multiple complete imputed datasets, analyzing each imputed dataset separately and combining the results via Rubin's rule [Rub04, Pat02, BWR13]. Capturing the uncertainty in the dataset is especially important in the medical setting, in which diagnostic, prognostic and treatment decisions must be made on the basis of the imputed values [Mac10, SWC09]. In our setting, we use dropout [SHK14] to produce multiple imputations; see Section 3.3.4.

To demonstrate the power of our method, we apply it to five different public real-world medical datasets: the MIMIC-III [JPS16] dataset, the clinical deterioration dataset used in [AYH18], the UNOS dataset for heart transplantation, the UNOS dataset for lung transplantation (both available at `https://www.unos.org/data/`), and the UK Biobank dataset [Pal07]. We show that our method yields large and statistically significant improvements in estimation accuracy over previous methods, including interpolation methods such as [KL12, MP10], imputation methods such as [Rub04, GSF10, WRW11, SB11], RNN-based imputation methods such as [CBS16, LKW16, CPC18] and matrix completion methods such as [CR09]. For the MIMIC-III and clinical deterioration datasets the patient measurements were made frequently (hourly basis), and our method provides Root Mean Squared Error (RMSE) improvement of more than 50% over all 11 benchmarks. For the UNOS heart and lung transplantation datasets and the Biobank dataset, the patient measurements were made much less frequently (yearly basis), but our method still provides RMSE improvement of more than 40% in most cases, and significant improvements in the other cases. We also show that this improvement in estimation yields (smaller) improvements in the predictions of outcomes (patients' future states). A number of experiments based on these same datasets show that the extent to which our method improves on outcomes depends on the method used for prediction, on the way in which our model is optimized in training, on the amount of data available (both in terms of the number of patients for whom we have data and on the amount of data available for each patient), and on the nature and extent of missing data. These results illustrate the important point that, as mentioned earlier, there are *many*

reasons for imputing missing data [BWR13, DCI16] – for the estimation of parameters (e.g. means or regression coefficients), for determination of confidence intervals and significance, as well as for prediction – and that no single method for imputing data can be expected to be superior *on all datasets* or *for all reasons*.

As [Men94] has emphasized, an extremely desirable aspect of any imputation method is that it be *congenial*; i.e. that it should produce imputed values in a manner that preserves the original relationships between features and labels. As we demonstrate using the complete Biobank dataset, our method is also more congenial than the best competing benchmarks; see Section 3.4.8.

3.1 Related works

As we have noted, there are three standard and very widely-used methods for dealing with missing data: interpolation, imputation and matrix completion. Interpolation methods [KL12, MP10] attempt to reconstruct missing data by capturing the temporal relationship *within* each data stream but not the relationships *across* streams. Imputation methods [Rub04, GSF10, WRW11, SB11] attempt to reconstruct missing data by capturing the synchronous relationships *across* data streams but not the temporal relationships *within* streams. Matrix completion methods [CR09, YRD16, SSS16] treat the data as static – ignoring the temporal aspect – or perfectly synchronized and assume a specific model of the data-generating process and/or the pattern of missing data.

There is also a substantial literature that uses Recurrent Neural Networks (RNNs) for prediction on the basis of time series with missing data. For example, [GB96] first replaces all the missing values with a mean value, then uses the feedback loop from the hidden states to update the imputed values and finally uses the reconstructed data streams as inputs to a standard RNN for prediction. [TB98] uses the Expectation-Maximization (EM) algorithm to impute the missing values and again uses the reconstructed data streams as inputs to a standard RNN for prediction. [PG02] uses a linear model to estimate missing values from the latest measurement and the hidden state within each stream followed by a standard

RNN for prediction. In the first two of these papers, missing values are imputed by using only the synchronous relationships across data streams but not the temporal relationships within streams; in the third paper, missing values are interpolated by using only the temporal relationships within each stream but not on the relationships across streams.

A more recent literature extends these methods to deal with both missing data and irregularly sampled data [CBS16, LKW16, CPC18, KJC17]. All of these papers use the sampling times to capture the informative missingness and time interval information to deal with irregular sampling, using the measurements, sampling information and time intervals as the inputs of an RNN. However, they differ in the replacements they use for missing values. [CBS16, LKW16, KJC17] replace the missing values with 0, mean values or latest measurements – all of which are independent of either the intra-stream or inter-stream relationships or both. [CPC18] imputes the missing values using only the most recent measurements, the mean value of each stream, and the time interval.

3.2 Problem formulation

Our formulation and method are applicable to a wide variety of settings with missing data. However, for ease of exposition – and to facilitate the discussion of our application to medical datasets – it is convenient to adopt medical terminology throughout.

We consider a dataset consisting of N patients. For each patient, we have a multivariate time series data stream of length T (the length T and the other components of the dataset may depend on the patient n but for the moment we suppress the dependence on n) that consists of time stamps $\mathcal{S}$, measurements $\mathcal{X}$, and labels $\mathcal{Y}$, sampled from an (unknown) underlying distribution $\mathcal{F}$: $(\mathcal{S}, \mathcal{X}, \mathcal{Y}) \sim \mathcal{F}$.

For each t the *time stamp* $s_t \in \mathbb{R}$ represents the *actual time* at which the measurements x_t were taken. For convenience we normalize so that $s_1 = 0$ (so that we are measuring actual times for each patient beginning from the first observation for that patient); we assume actual times are strictly increasing: $s_{t+1} > s_t$ where $0 \leq t < T$. Note that the measurements may not be sampled regularly, so that the interval $s_{t+1} - s_t$ between successive measurements

need not be constant.

There are D streams of measurements. We view each measurement as a real number, but it will typically be the case that not every stream is actually observed/measured at s_t. Hence we adopt notation in which the set of possible measurements at the t-th time stamp s_t is $\mathbb{R}_* = \mathbb{R} \cup \{*\}$. We interpret $x_t^d = *$ to mean that the stream d was not measured at s_t; otherwise $x_t^d \in \mathbb{R}$ is the actual measurement of stream d at s_t. (In computations with neural networks, we set $x_t^d = 0$ when the measurement x_t^d is missing. This guarantees that the missing measurement has no effect on the architecture.) For convenience, we scale all measurements to lie in the interval $[0, 1]$.

It is convenient to introduce some additional notation. For each t, define the index m_d^t to equal 0 if $x_t^d = *$ (i.e. the stream d was not measured at s_t) and to equal 1 if $x_t^d \in [0, 1]$ (the stream d was measured at s_t). We define δ_t^d to be the actual amount of time that has elapsed from s_t since the stream d was measured previously; δ_t^d can be defined by setting $\delta_1^d = 0$ and then proceeding recursively as follows:

$$\delta_t^d = \begin{cases} s_t - s_{t-1} + \delta_{t-1}^d & \text{if } t > 1, m_{t-1}^d = 0. \\ s_t - s_{t-1} & \text{if } t > 1, m_{t-1}^d = 1 \end{cases}$$

Write $\boldsymbol{\delta}_t$ for the vector of elapsed times at time stamp t and $\Delta = \{\boldsymbol{\delta}_1, \boldsymbol{\delta}_2, ..., \boldsymbol{\delta}_T\}$.

The label y_t represents the outcome realized at time stamp t (actual time s_t) such as discharge, clinical deterioration, death. $\mathcal{Y}$ is the vector of outcomes for this patient. Again, we scale so the labels (and eventually predictions) lie in the interval $[0, 1]$. Frequently the outcome is binary in which case $y_t = 0$ or $y_t = 1$.

The information available for a particular patient n is therefore a triple consisting of a sequence of time stamps, an array of measurements at each time stamp (with the above convention about missing measurements), and an array of labels at each time stamp. It is convenient to use functional notation to identify information about a particular patient, so $x_t^d(n)$ is the measurement of stream d at time stamp t for patient n, etc. The entire dataset consists of all the triples for all the patients $\mathcal{D} = \{(\mathcal{S}(n), \mathcal{X}(n), \mathcal{Y}(n)\}_{n=1}^N$.

Our objective is to find a function $\mathbf{f}$ that provides the best estimate of missing values; i.e. the estimate that minimizes the estimation loss. As is usually done, we measure loss as the squared error, so if x_t^d is an (unobserved) actual measurement (sampled from $\mathcal{F}$) and $\hat{x}_t^d = f_t^d(\mathcal{S}, \mathcal{X})$ is the estimate formed on the basis of observed data, then the squared loss for this particular measurement is $\mathcal{L}(\hat{x}_t^d, x_t^d) = (\hat{x}_t^d - x_t^d)^2$. Hence the formal optimization problem is to find a function $\mathbf{f}$ to solve:

$$\min_{\mathbf{f}} \mathbb{E}_{\mathcal{F}}\Big[\sum_{t=1}^{T}\sum_{d=1}^{D}(1-m_t^d)\mathcal{L}(\hat{x}_t^d, x_t^d)\Big] = \min_{\mathbf{f}} \mathbb{E}_{\mathcal{F}}\Big[\sum_{t=1}^{T}\sum_{d=1}^{D}(1-m_t^d)(f_t^d(\mathcal{S}, \mathcal{X}, \mathcal{Y}) - x_t^d)^2\Big]. \quad (3.1)$$

Note that the function $\mathbf{f}$ we seek depends on the particular d and t, and on the entire array of time stamps and measurements – but not on labels (which may not be observed). Also note that the formal problem asks to find an $\mathbf{f}$ that minimizes the loss with respect to the *true distribution*. Of course we do not observe the true distribution and cannot compute the true loss, so we will minimize the empirical loss.

3.3 Multi-directional Recurrent Neural Networks (M-RNN)

Suppose that stream d was not measured at time stamp t, so that $x_t^d = *$. We would like to form an estimate $\hat{x}_t^d$ of what the actual measurement would have been. As we have noted, familiar interpolation methods use only the measurements $x_{t'}^d$ of the fixed data stream d for other time stamps $t' \neq t$ (perhaps both before and after t) – but ignore the information contained in other data streams $d' \neq d$; familiar imputation methods use only the measurements $x_t^{d'}$ at the fixed time t for other data streams $d' \neq d$ – but ignores the information contained at other times $t' \neq t$. Because information is often correlated both *within and across* data streams, each of these familiar approaches throws away potentially useful information. Our approach forms an estimate $\hat{x}_t^d$ using measurements both *within the given data stream and across other data streams*. In principle, we could try to form the estimate $\hat{x}_t^d$ by using *all* the information in $\mathcal{D}$. However, this would be impractical because it would require learning a number of parameters that is on the order of the square of the number

of data streams, and also because it would create a serious danger of over-fitting. Instead, we propose an efficient hierarchical learning framework using a novel RNN architecture that effectively allows us to capture the correlations both within streams and across streams. Our approach limits the number of parameters to be learned to be of the linear order of the number data streams and avoids over-fitting. See Fig. 3.1.

Our basic single-imputation M-RNN consists of 2 blocks: an Interpolation block and an Imputation block; see Fig. 3.2. (Our construction puts the Imputation block after the Interpolation block in order to use the outputs of the Interpolation block to improve the accuracy of the Imputation block; as we discuss later, it would not be useful to put the Interpolation block after the Imputation block.) To produce multiple imputations, we adjoin an additional dropout layer to the basic single-imputation M-RNN. (We defer the details until Section 3.3.4.) The entire source codes of M-RNN implementation are publicly available in the following link: `http://github.com/jsyoon0823/MRNN/`.

3.3.1 Error/Loss

As formalized above in Equation (3.1), our overall objective is to minimize the error that would be made in estimating missing measurements. Evidently, we cannot estimate the error of a measurement that was not made and hence is truly missing in the dataset. Instead we fix a measurement x_t^d that *was* made and is present in the dataset, form an estimate $\hat{x}_t^d$ for x_t^d using only the dataset with x_t^d removed (which we denote by $\mathcal{D} - x_t^d$), and then compute the error between the estimate $\hat{x}_t^d$ and the actual measurement x_t^d. As above, we use the squared error $(\hat{x}_t^d - x_t^d)^2$ as the loss for this particular estimate; as the total loss/error for the entire dataset $\mathcal{D}$ we use the *mean squared error* (MSE):

$$\mathcal{L}(\hat{\mathbf{x}}, \mathbf{x}) = \sum_{n=1}^{N} \left[\frac{\sum_{t=1}^{T_n} \sum_{d=1}^{D} m_t^d(n) \times (\hat{x}_t^d(n) - x_t^d(n))^2}{\sum_{t=1}^{T_n} \sum_{d=1}^{D} m_t^d(n)} \right]$$

Note that this is the empirical error, which only utilized actually achievable variables.

3.3.2 Interpolation block

The Interpolation block constructs an interpolation function $\mathbf{\Phi}_d$ that operates *within* the d-th stream. To emphasize that the output $\tilde{x}_t^d$ of the interpolation block depends only on the d-th data stream with x_t^d removed, we write $\tilde{x}_t^d = \mathbf{\Phi}_d(\mathcal{D}^d - x_t^d)$, where $\mathcal{D}^d$ is the d-th stream of the entire dataset $\mathcal{D}$, and the notation $\mathcal{D}^d - x_t^d$ emphasizes that we have removed x_t^d. It is important to keep in mind that the construction uses only the data from stream d, not the data from other streams. We construct $\mathbf{\Phi}_d$ using a bi-directional recurrent neural network (Bi-RNN). However, unlike a conventional Bi-RNN [GS05], the timing of inputs into the hidden layer is lagged in the forward direction and advanced in the backward direction: at t, inputs of forward hidden states come from $t-1$ and inputs of backward hidden states come from $t+1$. (This procedure ensures that the actual value x_t^d is not used in the estimation of $\tilde{x}_t^d$.) Note that each data stream uses its own Bi-RNN architecture ($\mathbf{\Phi}_d$). The inputs of the Interpolation block consist of the feature vector x, the mask vector m, and the elapsed time vector δ (defined in Section 3.2, and extracted from the original data streams). If we write $\mathbf{z}_t^d = [x_t^d, m_t^d, \delta_t^d]$ (note that we explicitly include δ_t^d as the additional input to deal with the irregular sampling procedures) then a more mathematical description is:

$$
\begin{aligned}
\tilde{x}_t^d &= g(U^d[\overrightarrow{\mathbf{h}}_t^d; \overleftarrow{\mathbf{h}}_t^d] + \mathbf{c}_o^d) = g(\overrightarrow{U}^d \overrightarrow{\mathbf{h}}_t^d + \overleftarrow{U}^d \overleftarrow{\mathbf{h}}_t^d + \mathbf{c}_o^d) \\
\overrightarrow{\mathbf{h}}_t^d &= (1 - \overrightarrow{\mathbf{u}}_t^d) \circ \overrightarrow{\mathbf{h}}_{t-1}^d + \overrightarrow{\mathbf{u}}_t^d \circ q(\overrightarrow{W}_h^d(\overrightarrow{\mathbf{r}}_t^d \circ \overrightarrow{\mathbf{h}}_{t-1}^d) + \overrightarrow{V}_h^d \mathbf{z}_{t-1}^d + \overrightarrow{\mathbf{c}}_h^d) \\
\overrightarrow{\mathbf{u}}_t^d &= \gamma(\overrightarrow{W}_u^d \overrightarrow{\mathbf{h}}_{t-1}^d + \overrightarrow{V}_u^d \mathbf{z}_{t-1}^d + \overrightarrow{\mathbf{c}}_u^d) \\
\overrightarrow{\mathbf{r}}_t^d &= \gamma(\overrightarrow{W}_r^d \overrightarrow{\mathbf{h}}_{t-1}^d + \overrightarrow{V}_r^d \mathbf{z}_{t-1}^d + \overrightarrow{\mathbf{c}}_r^d) \\
\overleftarrow{\mathbf{h}}_t^d &= (1 - \overleftarrow{\mathbf{u}}_t^d) \circ \overleftarrow{\mathbf{h}}_{t+1}^d + \overleftarrow{\mathbf{u}}_t^d \circ q(\overleftarrow{W}_h^d(\overleftarrow{\mathbf{r}}_t^d \circ \overleftarrow{\mathbf{h}}_{t+1}^d) + \overleftarrow{V}_h^d \mathbf{z}_{t+1}^d + \overleftarrow{\mathbf{c}}_h^d) \\
\overleftarrow{\mathbf{u}}_t^d &= \gamma(\overleftarrow{W}_u^d \overleftarrow{\mathbf{h}}_{t+1}^d + \overleftarrow{V}_u^d \mathbf{z}_{t+1}^d + \overleftarrow{\mathbf{c}}_u^d) \\
\overleftarrow{\mathbf{r}}_t^d &= \gamma(\overleftarrow{W}_r^d \overleftarrow{\mathbf{h}}_{t+1}^d + \overleftarrow{V}_r^d \mathbf{z}_{t+1}^d + \overleftarrow{\mathbf{c}}_r^d)
\end{aligned}
$$

(As can be seen from these equations, we are using a bidirectional GRU.) Here, g, q, γ are activation functions. (In principle, any activation functions, such as Rectified Linear Unit (ReLU), tanh, etc., could be used; here we use ReLU.) The arrows indicate forward/backward

direction and $\circ$ indicates element-wise multiplication. As we have emphasized, in this interpolation block, we are only using/capturing the temporal correlation *within each data stream*. In particular, the parameters for each data stream are learned separately, and the number of parameters that must be learned is linear in the number of streams D. Note that $\tilde{x}_t^d$ is not the final output of our M-RNN architecture and is not necessarily an estimate of x_t^d.

3.3.3 Imputation block

The Imputation blocks constructs an imputation function $\mathbf{\Psi}$ that operates *across* streams. To again emphasize that the estimate $\hat{x}_t^d$ for x_t^d depends on the data with x_t^d removed, we write $\hat{x}_t^d = \mathbf{\Psi}(\mathcal{D}_t - x_t^d)$; again, keep in mind that now we are using only data at time stamp s_t, not data from other time stamps. ($\mathcal{D}_t$ represents the t-th time stamp of the entire dataset $\mathcal{D}$.) We construct the function $\mathbf{\Psi}$ to be independent of t, so we use fully connected layers; see the Imputation component of Fig 3.2. If we write $\mathbf{z}_t = [\tilde{\mathbf{x}}_t, \mathbf{m}_t]$ then a more mathematical description is:

$$\hat{\mathbf{x}}_t = \sigma(W\mathbf{h}_t + \alpha) \qquad\qquad \mathbf{h}_t = \phi(U\mathbf{x}_t + V\mathbf{z}_t + \beta)$$

where σ, ϕ are activation functions. It is important to keep in mind that the diagonal entries of U are zero and the off-diagonal entries of W are zero (i.e. W is diagonal) so that we do not use x_t^d in the estimation of $\hat{x}_t^d$.

We learn the functions $\{\mathbf{\Phi}_d\}_{d=1}^D$ and $\mathbf{\Psi}$ *jointly* using the stacked networks of Bi-RNN and Fully Connected (FC) layers, using MSE as the objective function.

$$\mathbf{\Psi}^*, \{\mathbf{\Phi}_d^*\}_{d=1}^D = \arg\min_{\mathbf{\Phi}_d, \mathbf{\Psi}} \mathcal{L}(\{\mathbf{\Psi}\Big(\{x_t^d, \mathbf{\Phi}_d\Big(\{x_\tau^d, m_\tau^d, \delta_\tau^d\}_{\tau=1}^T\Big), m_t^d\}_{d=1}^D\Big)\}_{t=1}^T, \mathbf{x}) \tag{3.2}$$

Note that $\tilde{x}_t$ is the output of the interpolation block, and $\hat{x}_t$ is the final output of the entire M-RNN architecture.

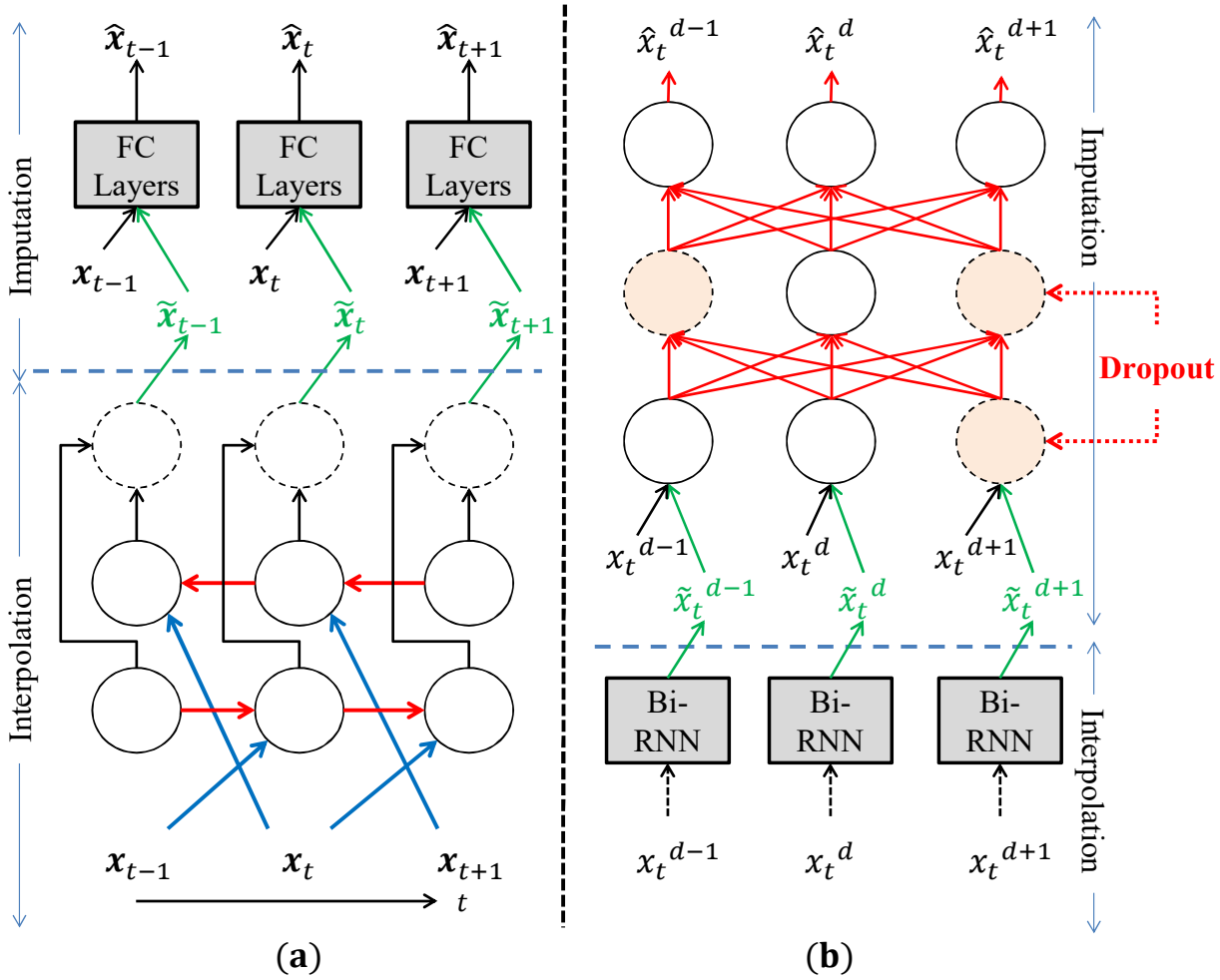

Figure 3.2: M-RNN Architecture. (a) Architecture in the time domain section; (b) Architecture in the feature domain section (Dropout is used for multiple imputations). Note that both $\tilde{\mathbf{x}}$ (the output of interpolation block) and $\mathbf{x}$ are inputs to the imputation block to construct $\hat{\mathbf{x}}$ (the output of imputation block).

3.3.4 Multiple imputations

It is well-understood that to account for the uncertainty in estimating missing values, it is useful to produce multiple estimates and generate multiple imputed datasets. These multiple imputed datasets can each be analyzed using standard methods and the results can be combined using Rubin's rule [Rub04]. In our case, we generate multiple imputed datasets using the well-known Dropout [SHK14] approach driven from the Bayesian Neural Network framework [GG16]: we randomly select neurons in the fully connected layers and delete those neurons and all their connections. (The dropout probability $p \in (0, 1)$ is a hyper-parameter to

be chosen; the neurons to be dropped are chosen according to the Bernoulli distribution with parameter p.) In the training stage, we conduct joint optimization (Equation (3.2)) using the dropout process. We then generate multiple outputs $\mathbf{o}_t$ by sampling different dropout vectors $\mathbf{R}$ from the Bernoulli distributions. This yields multiple imputations (MI). (To construct a single imputation (SI) we proceed in precisely the same way but set the dropout probability to 0. For comparisons, we normalize the final output by multiplying by p.)

3.3.5 Overall structure and computation complexity

We refer to the entire structure above as a *Multi-directional Recurrent Neural Network (M-RNN)*. We use the notations M-RNN (MI) and M-RNN (SI) to clarify whether we are producing multiple or single imputations. The entire training times for both M-RNN (MI) and M-RNN (SI) are less than 2 hours for all 6 datasets (described in Section 3.4.1) on a computer with Intel Core i7-4770 (3.4GHz) CPU with 32 GB RAM. With the same machine, the entire training time for Multiple Imputation with Chained Equation (MICE) [WRW11] is around 11 hours for all 6 datasets.

3.4 Results and discussions

3.4.1 Datasets

To evaluate the proposed M-RNN, we use five medical datasets: (1) MIMIC-III [JPS16], (2) Deterioration [AYH18], (3, 4) UNOS-Heart and UNOS-Lung from the UNOS (United Network for Organ Transplantation) dataset (available at `https://www.unos.org/data/`), (5) UK Biobank [Pal07]. The characteristics of which are summarized in Table 3.1.

3.4.2 Imputation accuracy on the given datasets

We begin by comparing the performance of our method (using both multiple imputations and single imputation) on the given datasets against 11 benchmarks with respect to the accuracy of imputing missing values. The benchmarks against which we compare are: the

Datasets	MIMIC-III	Deterioration	UNOS-Heart	UNOS-Lung	Biobank
# of Patients	23,160	6,094	69,205	32,986	3,902
# of Dimensions (Cont, Cat)	40 (31, 9)	38 (16, 22)	34 (10, 24)	34 (10, 24)	113 (67, 46)
Label ($y = 1$)	1,320 (5.7%)	306 (5.3%)	4,844 (7.0%)	2,276 (6.9%)	195 (5.0%)
Avg # of samples	24.3	34.3	6.2	4.0	3.0
Avg missing rate	75.0%	61.4%	59.1%	58.5%	0.0%
Avg measure Freq.	1 hr / 12 hrs	4 hrs / 24 hrs	1 year	1 year	2.3 years
Avg Corr within streams	0.4122	0.3436	0.1213	0.1157	0.2424
Avg Corr across streams	0.3127	0.3454	0.0875	0.0897	0.0506

Table 3.1: Summary of the datasets (Cont: Continuous, Cat: Categorical, Avg: Average, #: Number, Corr: Correlation, Freq: Frequency)

algorithms proposed in [CBS16, LKW16, CPC18]; Spline and Cubic Interpolation [KL12]; MICE [WRW11]; MissForest [SB11]; EM [GSF10]; the matrix completion algorithm of [CR09]; the Auto-Encoder algorithm proposed in [GW18]; and the Markov chain Monte Carlo (MCMC) method [Sch08]. As is common, we use root mean squared error (RMSE) as the measure of performance. In each experiment, we use 5-fold cross-validation.

In all of our experiments, we set the depth of the networks for M-RNN and for other neural network benchmarks (including RNN-based benchmarks and auto-encoder) to 4. (In the case of M-RNN, the interpolation block uses 2 layers and the imputation block uses 2 layers.) For M-RNN, there are 4 hidden nodes in each layer in the interpolation block and D hidden nodes in each layer in the imputation block. For the benchmarks, in order to make a fair comparison, we adjusted the number of hidden nodes in each layer to match the model capacity (the number of parameters for all models) of M-RNN. The number of batches is 64 for both M-RNN and benchmarks.

Table 3.2 shows the mean RMSE for our method and benchmarks, and the percentage improvement of RMSE for M-RNN (MI) over the benchmarks. (Note that we are unable to provide results for the EM algorithm on the UNOS-Heart and UNOS-Lung datasets because – at least for the implementation we use – the EM algorithm requires at least one patient for

Algorithm	Mean RMSE (% Gain of M-RNN (Multiple Imputations))				
	MIMIC-III	Deterioration	UNOS-Heart	UNOS-Lung	Biobank
M-RNN (MI)	**0.0141 (-)**	**0.0105 (-)**	**0.0479 (-)**	**0.0606 (-)**	**0.0637 (-)**
M-RNN (SI)	**0.0144 (-)**	**0.0108 (-)**	**0.0477 (-)**	**0.0609 (-)**	**0.0629 (-)**
[CBS16]	0.0337 (58.2%)	0.0258 (59.3%)	0.1352 (64.6%)	0.1343 (54.9%)	0.0812 (21.6%)
[LKW16]	0.0295 (52.2%)	0.0241 (56.4%)	0.1179 (59.4%)	0.1264 (52.1%)	0.0801 (20.5%)
[CPC18]	0.0292 (51.7%)	0.0233 (54.9%)	0.1057 (54.7%)	0.1172 (48.3%)	0.0778 (18.1%)
Spline	0.0735 (80.8%)	0.0215 (51.2%)	0.1102 (56.5%)	0.1199 (49.5%)	0.0845 (24.6%)
Cubic	0.0279 (49.5%)	0.0223 (52.9%)	0.1072 (55.3%)	0.1177 (48.5%)	0.0887 (28.2%)
MICE	0.0611 (76.9%)	0.0319 (67.1%)	0.1147 (58.2%)	0.1151 (47.4%)	0.0915 (30.4%)
MissForest	0.0293 (51.9%)	0.0264 (60.2%)	0.0489 (2.0%)	0.0652 (7.1%)	0.0892 (28.6%)
EM	0.0467 (69.8%)	0.0355 (70.4%)	-	-	0.0978 (34.9%)
Matrix Completion	0.0311 (54.7%)	0.0264 (60.2%)	0.0974 (50.8%)	0.0942 (35.7%)	0.0886 (28.1%)
Auto-encoder	0.0412 (66.0%)	0.0309 (65.0%)	0.0589 (18.7%)	0.0712 (14.9%)	0.0805 (20.9%)
MCMC	0.0437 (67.7%)	0.0364 (71.2%)	0.1091 (56.1%)	0.1124 (46.1%)	0.0936 (31.9%)

Table 3.2: Performance comparison for missing data estimation

whom data is complete, and the UNOS-Heart and UNOS-Lung datasets do not contain any such patient.)

As can be seen in Table 3.2, M-RNN achieves better performance (smaller RMSE) than all of the benchmarks on all of the datasets (for all comparisons are possible). With a single exception (the comparison with MissForest on the UNOS-Lung dataset) the performance improvements are statistically significant at the 95% level (i.e., $p < 0.05$), and many of the improvements are very large. For instance, for the Deterioration dataset, M-RNN using multiple imputations achieves RMSE of 0.0105 (95% CI: 0.0071-0.0138), while the *best* benchmark (Spline interpolation) achieves RMSE of 0.0215 (95% CI: 0.0178-0.0255); this represents an improvement of 51.2%.

The performance comparisons across datasets are revealing, if not necessarily surprising. The interpolation benchmarks (such as Spline, Cubic and RNN-based methods) work best on datasets, such as MIMIC-III and Deterioration, for which measurements were more frequent (and more highly correlated within each stream (see Table 3.1)); the imputation benchmarks work best on datasets, such as UNOS-Heart and UNOS-Lung, for which measurements were less frequent but for which there were many streams of data (many dimensions). The improvement of our method over all benchmarks is larger for the MIMIC-III and Deterioration

datasets because those datasets have many streams of frequently sampled data, so that our method gains a great deal from exploiting both the correlations within each data stream and the correlations across data streams. Conversely, the improvement of our method is smaller for the UNOS-Heart and UNOS-Lung datasets, because streams in those datasets are infrequently sampled to that there is less to be gained by exploiting the correlations within data streams. (The performances of the benchmarks for the Biobank dataset are mixed, and don't quite fit this same pattern, perhaps because Biobank is a small dataset (less than 4,000 patients with complete temporal data streams).)

3.4.2.1 Multiple imputations vs. Single imputation

As we have noted, the purpose of conducting multiple imputations is to reduce uncertainty/shrink confidence intervals (rather than to improve average performance). As is illustrated in the box-plot in Fig. 3.3(a) which shows the comparison of M-RNN with multiple imputations and M-RNN with a single imputation against the best benchmark (Cubic interpolation) on the MIMIC-III dataset, our multiple imputations do achieve this purpose. (For discussion of Fig. 3.3(b), which illustrates the corresponding reduction in uncertainty for prediction, see below.)

3.4.2.2 Combining models of interpolation and imputation

As we have already discussed, standard interpolation algorithms cannot capture the patterns across streams and standard imputation algorithms cannot capture the patterns within the streams. However, it is possible to combine a standard interpolation algorithm and standard imputation algorithm in an attempt to capture *both patterns*, and it might be thought that such a combination would be a fairer benchmark against which to compare our method. To put this idea to the test, we create a family of "joint algorithms" by first using an interpolation algorithm to interpolate the missing values, and then using the interpolated values as the initial points of an imputation algorithm to provide final imputed values. For this exercise, we use two standard interpolation methods (Cubic and Spline), and two standard imputation

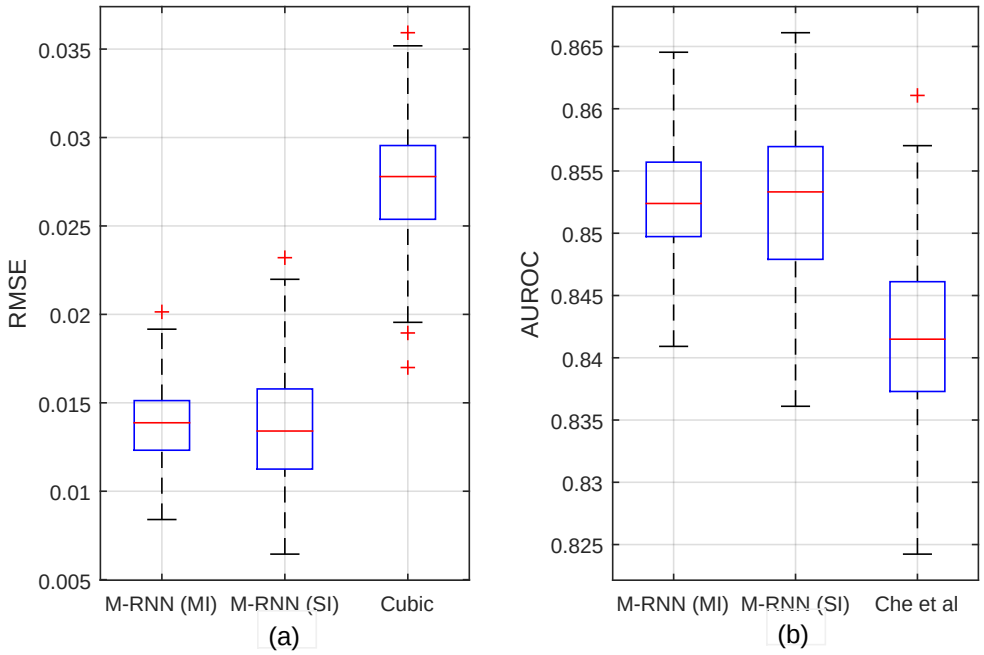

Figure 3.3: Box-plot comparisons between M-RNN (MI), M-RNN (SI) and the best benchmark. (a) RMSE comparison using MIMIC-III dataset, (b) AUROC comparison using MIMIC-III dataset. Red crosses represents outliers.

methods (MICE and MissForest) so that we have 4 interpolation-imputation combination models: Cubic + MICE, Cubic + MissForest, Spline + MICE, and Spline + MissForest.

Algorithm	**Mean RMSE (% Gain from Imputation Algorithm)**				
	MIMIC-III	Deterioration	UNOS-Heart	UNOS-Lung	Biobank
Spline + MICE	0.0602 (1.5%)	0.0320 (-0.3%)	0.1141 (0.5%)	0.1133 (1.7%)	0.0895 (2.2%)
Spline + MissForest	0.0291 (0.7%)	0.0259 (1.9%)	0.0491 (-0.4%)	0.0641 (1.4%)	0.0879 (4.1%)
Cubic + MICE	0.0605 (1.0%)	0.0315 (1.3%)	0.1137 (0.9%)	0.1138 (1.1%)	0.0901 (1.6%)
Cubic + MissForest	0.0289 (1.4%)	0.0261 (1.1%)	0.0493 (-0.8%)	0.0643 (1.4%)	0.0887 (3.2%)

Table 3.3: Performance comparison for joint interpolation/imputation algorithms

As Table 3.3 shows, however, the performances of these interpolation-imputation combination models are very similar to those of the performance of the simple imputation model that is used. Indeed, the largest RMSE performance improvement is only 0.0018. The reason for this is that imputation methods use algorithms that operate iteratively until they converge, so that their performance is rather robust to the initialization. Hence, although the interpolation part of the joint models captures some of the inter-stream information, the iterative imputation part ignores most of what is captured.

3.4.3 Source of gains

As illustrated in Fig. 3.2, our M-RNN consists of an Interpolation block and an Imputation block. To understand where the gains of our approach come from, we compare the performance of that is achieved when we use only the Interpolation block or only the Imputation block; the results are shown in Table 3.4.

Datasets	**M-RNN (Mean RMSE; % Gain)**		
	Only Interp	**Only Impute**	**Interp + Impute**
MIMIC-III	0.0191 (26.2 %)	0.0312 (54.8 %)	0.0141 (-)
Deterioration	0.0133 (21.1 %)	0.0295 (64.4 %)	0.0105 (-)
UNOS-Heart	0.0897 (46.6 %)	0.0531 (9.8 %)	0.0479 (-)
UNOS-Lung	0.0998 (39.3 %)	0.0734 (17.4 %)	0.0606 (-)
Biobank	0.0794 (19.8 %)	0.0778 (18.1 %)	0.0637 (-)

Table 3.4: Source of Gain of M-RNN. (Performance degradation from original M-RNN)

The Interpolation block is intended to exploit the correlations within each data stream and the Imputation block is intended to exploit the correlations across streams, so it is to be expected that the largest gains of our M-RNN method should come from the Interpolation block for the datasets (MIMIC-III and Deterioration) which are frequently sampled and have large temporal correlations, and should come from the Imputation block for the datasets (UNOS-Heart and UNOS-Lung) which are infrequently sampled but have many data streams. As shown in Table 3.4, these intuitions are indeed supported by the experiments.

3.4.4 Additional experiments

The experiments we have described above demonstrate that our method significantly outperforms a wide variety of benchmarks for the imputation of missing data on five somewhat representative datasets. However it is natural to ask how our method would compare in other circumstances. To get some understanding of this, we conducted four sets of experiments based on the MIMIC-III dataset: increasing the amount of missing data, reducing the number of data streams, reducing the number of samples, and reducing the number of measurements

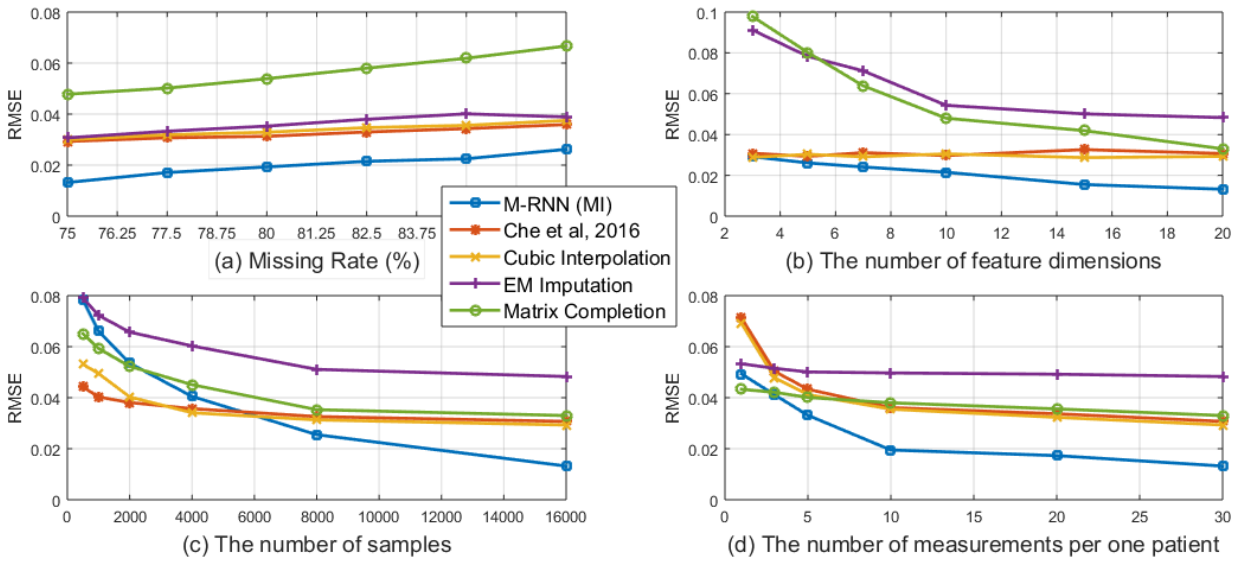

Figure 3.4: Imputation accuracy for the MIMIC-III dataset with various settings (a) Additional data missing at random, (b) Feature dimensions chosen at random, (c) Samples chosen at random, (d) Measurements chosen at random

per patient. Within each set of experiments, we conducted 10 trials for each value of the parameter being studied (e.g. amount of missing data), and we report the average over these 10 trials. The results are described below and in Fig. 3.4. Although the results of these experiments are extremely suggestive, we caution the reader that these are only a specific set of experiments and that one should be careful about drawing general conclusions.

3.4.4.1 Amount of missing data (Fig. 3.4 (a))

To evaluate the performance of M-RNN in comparison to benchmarks in settings with more missing data, we constructed sub-samples of the MIMIC-III dataset by randomly removing 10%, 20%, 30%, 40%, 50% of the actual data and carrying out the same estimation exercise as above on the smaller datasets that remain. (Recall that in the original MIMIC-III dataset, 75% of the data is already missing; hence removing 50% of the data present leads to an artificial dataset in which 87.5% of the data is missing.) The graph in Fig. 3.4(a) shows the performance of M-RNN against the *best* benchmarks of each type for these smaller datasets. As can be seen, M-RNN continues to substantially outperform the benchmarks. Note that as the amount of missing data increases the improvement of M-RNN over the imputation

benchmark(s) increases, but the improvement over the interpolation benchmarks decreases.

3.4.4.2 Number of data streams (Fig. 3.4 (b))

As we have noted, typical medical datasets contain many data streams (many feature dimensions). To evaluate the performance of M-RNN in comparison to benchmarks in settings with fewer data streams, we conducted experiments in which we reduced the number of data streams (feature dimensions) of MIMIC-III. In the original MIMIC-III dataset the number of data streams is $D = 40$; we conducted experiments with $D = 3, 5, 7, 10, 15, 20$ data streams. (In each case, we conducted 10 trials in which we selected data streams at random; we report the average of these 10 trials.) As expected, the performance of M-RNN degrades when there are fewer data streams, but as Fig. 3.4(b) shows, M-RNN still outperforms the benchmarks. (Note that interpolation methods are insensitive to the number of data streams because they operate only *within each data stream separately*.)

3.4.4.3 Number of samples (Fig. 3.4 (c))

The original MIMIC-III dataset has $N = 23,160$ samples (patients). To understand the performance of M-RNN in comparison to benchmarks in settings with fewer samples, we conducted experiments in which we used only subsets of all patients (samples) of sizes $N = 500, 1000, 2000, 4000, 8000, 16000$. Because M-RNN has to learn many parameters, it should come as no surprise that, as Fig. 3.4(c) shows, the performance of M-RNN degrades badly – and indeed is worse than that of (some) other benchmarks – when the number of samples is too small, but M-RNN outperforms all the benchmarks as soon as the number of training samples exceeds $N = 7,000$. (However, one should not necessarily take the figure $N = 7,000$ as representing a cut-off below which M-RNN should not be applied, because M-RNN outperforms the benchmarks on the Deterioration and Biobank datasets, which contain only 6,094 samples and 3,902 samples, respectively.)

3.4.4.4 Number of measurements per patient (Fig. 3.4 (d))

We have already noted that, in our datasets, MIMIC-III and Deterioration have many (relatively frequent) measurements per patient, while the other datasets have only a few (and infrequent) measurements per patient and that this leads to differences in performance of M-RNN. To further explore this effect, we created subsets of the MIMIC-III dataset with $T = 1, 3, 5, 10, 20, 30$ measurements per patient. As might be expected, and as Fig. 3.4(d) shows, having fewer measurements per patient degrades the performance of interpolation-based algorithms but has little effect on pure imputation-based methods; the performance of M-RNN is also degraded, but to a much lesser extent.

3.4.5 Prediction accuracy

As we have noted, there are many reasons for imputing missing data; one such is to improve predictive performance. We therefore compare our method against the same 11 benchmarks with respect to the accuracy of predicting labels. (See the description of the datasets in Section 3.4.1 for labeling in each case.) For this purpose, we use Area Under the Receiver Operating Characteristic Curve (AUROC) as the measure of performance. To be fair to all methods of imputing missing values, we use the same predictive model (a simple 1-layer RNN) in all cases.

3.4.5.1 Prediction accuracy on the original datasets

In this subsection, we evaluate the effects of the imputations on the prediction of labels (outcomes), which in the cases at hand correspond to prognoses.

Table 3.5 shows the mean and percentage performance gain of M-RNN (MI) in comparison with the benchmarks on all the datasets. M-RNN – which we have already shown to achieve the best imputation accuracy – also yields the best prediction accuracy. However, even in cases where the improvement in imputation accuracy is large and statistically significant, the improvements in prediction accuracy are sometimes smaller and not always statistically

Algorithm	AUROC (Gain of M-RNN (MI) as % improvement in 1-AUROC)				
	MIMIC-III	Deterioration	UNOS-Heart	UNOS-Lung	Biobank
M-RNN (MI)	**0.8531 (-)**	**0.7779 (-)**	**0.6855 (-)**	**0.6762 (-)**	**0.8955 (-)**
M-RNN (SI)	**0.8530 (-)**	**0.7783 (-)**	**0.6858 (-)**	**0.6759 (-)**	**0.8948 (-)**
[CBS16]	0.8381 (9.3%)	0.7558 (9.0%)	0.6505 (10.0%)	0.6557 (6.0%)	0.8802 (12.8%)
[LKW16]	0.8402 (8.1%)	0.7551 (9.3%)	0.6574 (8.2%)	0.6561 (5.8%)	0.8748 (16.5%)
[CPC18]	0.8410 (7.6%)	0.7593 (7.7%)	0.6583 (8.0%)	0.6520 (7.0%)	0.8826 (11.0%)
Spline	0.8407 (7.8%)	0.7542 (9.6%)	0.6477 (10.7%)	0.6520 (7.0%)	0.8731 (17.7%)
Cubic	0.8397 (8.4%)	0.7569 (8.6%)	0.6468 (11.0%)	0.6517 (7.0%)	0.8643 (23.0%)
MICE	0.8377 (9.5%)	0.7571 (8.6%)	0.6397 (12.7%)	0.6509 (7.2%)	0.8850 (9.1%)
MissForest	0.8368 (10.0%)	0.7578 (8.3%)	0.6740 (3.5%)	0.6587 (5.1%)	0.8767 (15.2%)
EM	0.8312 (13.0%)	0.7531 (10.0%)	-	-	0.8794 (13.3%)
Matrix Completion	0.8401 (8.1%)	0.7551 (9.3%)	0.6712 (4.3%)	0.6579 (5.3%)	0.8865 (7.9%)
Auto-encoder	0.8399 (8.2%)	0.7488 (11.6%)	0.6633 (6.6%)	0.6574 (5.5%)	0.8785 (14.0%)
MCMC	0.8298 (13.7%)	0.7512 (10.7%)	0.6417 (12.2%)	0.6512 (7.2%)	0.8667 (21.6%)

Table 3.5: Performance comparison for patient state prediction with a 1-layer RNN (Performance gain is computed in terms of 1-AUROC)

significant. For instance, on the Deterioration dataset, the AUROC of M-RNN (MI) is 0.7779 (95% CI: 0.7678-0.7868); the best benchmark is [CPC18] with AUROC of 0.7593 (95% CI: 0.7478-0.7702). Similarly, on the UNOS-Heart dataset, the AUROC of M-RNN (MI) is 0.6855 (95% CI: 0.6781-0.6913); the best benchmark is MissForest, with AUROC of 0.6740 (95% CI: 0.6651-0.6817).

It should be noted that, by using mean squared error as the loss function, we have deliberately optimized M-RNN for *imputation accuracy*. If we want to optimize M-RNN for *prediction accuracy* we might do better by using a different loss function, such as cross-entropy.

3.4.6 Prediction accuracy with various missing rates

As discussed above, we carried out experiments with increased rates of missing data in order to understand the implications for the accuracy of imputation. We also carried out experiments with increased rates of missing data in order to understand the implications for the accuracy of prediction. To explore the predictive performance for a wide range of missing rates (from 0% to 90%), we begin with the Biobank dataset, which is a complete dataset. We randomly remove 10% to 90% of the measurements (with increments of 10%)

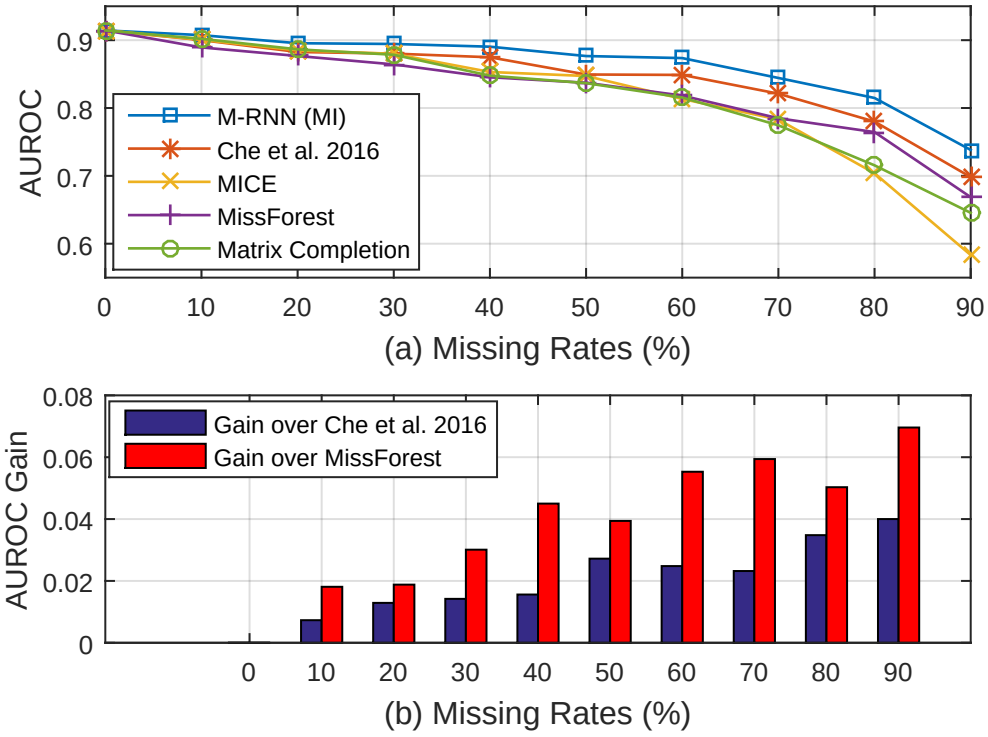

Figure 3.5: (a) The AUROC performance with various missing rates, (b) The AUROC gain over the two most competitive benchmarks

to create multiple datasets with different missing rates. (In each case we use 80% of the data for training and 20% for testing.) As before, we use M-RNN and various benchmarks for imputing missing data and a 1-layer RNN as the predictive model. (In this setting we are predicting a clinical diagnosis of diabetes.) In each setting, we conducted 10 trials, and report the performance in terms of AUROC.

Fig. 3.5 (a) illustrates the impact (in terms of AUROC) of increasing amounts of missing data for M-RNN and various benchmarks. As Fig. 3.5 (a) shows, for M-RNN and all benchmarks, the prediction performance decreases as the amount of missing data increases. However, as Fig. 3.5 (b) shows, M-RNN continues to outperform the benchmarks; indeed, the performance gap between M-RNN and the benchmarks widens when more data is missing. That is: the importance of accurate imputation is greater when more data is missing.

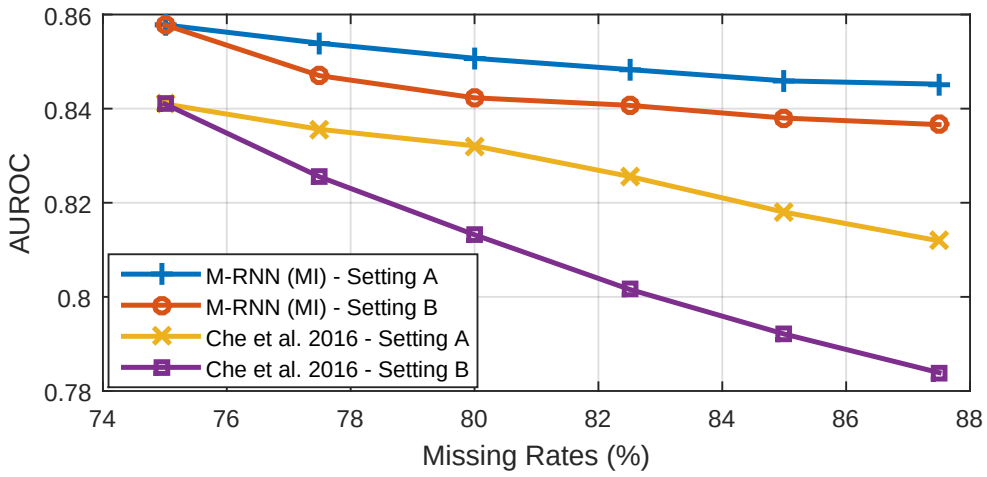

Figure 3.6: AUROC comparisons in Settings A and B using MIMIC-III dataset

3.4.7 The importance of specific features

To this point, we have treated all missing data as equally important and given the same weight to all errors. However, this is not always the right thing to do. In particular, it is clear that not all missing data is equally important for prediction. To understand the importance of missing data for purposes of prediction we conduct two experiments in parallel. For the first experiment (which we call Setting A: Purely Random Removal), we construct 5 sub-samples of the MIMIC-III dataset by randomly removing an *additional* 10%, 20%, 30%, 40%, 50% of the measurements for randomly chosen features. For the second experiment (which we call Setting B: Correlated Random Removal) we first identify the four features that are most highly correlated with the mortality label; those are anion gap, bicarbonate, systolic blood pressure, and potassium. We then construct 5 sub-samples of the MIMIC-III dataset by removing an *additional* 10%, 20%, 30%, 40%, 50% of the measurements for *these specific features*. In both cases we repeat the exercise 10 times and report average results. We then compare the prediction performance of M-RNN (MI) with the best benchmarks; the results are shown visually in Fig. 3.6.

Fig. 3.6 shows that M-RNN outperforms the best benchmarks for every sub-sample and the improvement in performance is greater for the sub-samples for which more data is missing. The improvement in performance is statistically significant (p-value < 0.05) when

an additional 30% or more of the measurements - i.e. a total of 82.5% of the measurements - (or of the most important features for Setting B) - are missing. In particular, the prediction performance of M-RNN is much less sensitive to the *amount of data that is missing* and to *which data is missing*.

3.4.8 Congeniality of the model

As [Men94] has emphasized, an extremely desirable aspect of any imputation method is that it produce imputed values in a manner that is consistent and preserves the original relationships between features and labels; [Men94] refers to this as *congeniality*. Congeniality of an imputation model can be evaluated with respect to a particular model of the feature-label relationships by computing the model parameters for the true complete data and the imputed data and measuring the difference between parameters according to some specified metric. Of course no imputation method can be expected to be perfectly congenial, but we argue that our method is more congenial – i.e. better preserves the relationships between features and labels – than benchmarks. To see this, we exploit the Biobank dataset; this is a complete dataset, so that it is possible to compare the relationship between the actual (original) data and labels and the relationship between the the imputed data and labels.

In our particular experiment, we delete 20% of the data and impute the missing data using our M-RNN and the 4 best benchmarks (the method of [CPC18], Cubic Interpolation, MissForest and Matrix Completion). As a model of the feature-label relationship, we use a logistic regression. As a metric of the difference between the logistic regression parameters $\mathbf{w}$ for the actual data and $\hat{\mathbf{w}}$ for the imputed data (which can be interpreted as a measure of the uncongeniality of the imputation) we report both the mean bias $\|\mathbf{w} - \hat{\mathbf{w}}\|_1$ and the root mean squared error $\|\mathbf{w} - \hat{\mathbf{w}}\|_2$.

As can be seen in Table 3.6, in comparison with the 4 best benchmarks, M-RNN achieves both smaller mean bias and small root mean squared error between the original and imputed representations of feature-label relationship. (With the exception of MissForest, all the performance improvements of M-RNN are statistically significant at the 95% level.) Thus

Algorithm	Mean Bias ($||\mathbf{w} - \hat{\mathbf{w}}||_1$)	Root Mean Squared Error($||\mathbf{w} - \hat{\mathbf{w}}||_2$)
M-RNN (MI)	0.0814 ± 0.0098	0.1229 ±0.0151
[CPC18]	0.1097 ± 0.0104	0.1649 ± 0.0212
Cubic Interpolation	0.1169 ± 0.01075	0.1816 ± 0.0201
MissForest	0.0842 ±0.0103	0.1312 ± 0.0139
Matrix Completion	0.1001 ± 0.0125	0.1551 ± 0.0230

Table 3.6: Congeniality of imputation models

our method is more congenial (to the logistic regression model) than the benchmarks.

3.4.9 M-RNN when data is missing at random

The experiments above are designed to demonstrate the superiority of the M-RNN framework in comparison to the benchmarks in settings where data is Missing Completely at Random (MCAR) [Rub04] but it is important to understand the comparison in the settings where data is Missing at Random (MAR) - but *not* Missing Completely at Random. In this subsection, we show that M-RNN also outperforms the benchmarks when data is Missing at Random (MAR). (The assumption that data is Missing at Random is standard in the medical setting.)

To accomplish this, we again begin with the complete Biobank dataset and remove 20% of the data. However in this case we do not remove data completely at random; rather, we use the following procedure.[1] Using induction, we define the probability that the component i of sample n at time t is observed, conditional on the missingness and values (if observed) of the previous $i-1$ components at time t [Rub04] to be

$$P_i^t(n) = \frac{p^m \cdot N \cdot e^{-\sum_{j<i} w_j m_j^t(n) x_j^t(n) + b_j (1 - m_j^t(n))}}{\sum_{l=1}^{N} e^{-\sum_{j<i} w_j m_j^t(l) x_j^t(l) + b_j (1 - m_j^t(l))}}$$

where p^m corresponds to the average missing rate (in our experiment, $p^m = 0.2$), and w_j, b_j are sampled from $\mathcal{U}(0,1)$ (but are only sampled once for the entire dataset). We sequentially sample $m_1^t, ..., m_D^t$ for each feature vector.

[1] Other procedures are certainly possible.

Algorithm	RMSE (% Gain of M-RNN (MI))	
	MCAR	**MAR**
M-RNN (MI)	**0.0637 (-)**	**0.1135 (-)**
[CPC18]	0.0778 (18.1%)	0.1243 (8.7%)
Cubic Interpolation	0.0887 (28.2%)	0.1278 (11.2%)
MissForest	0.0892 (28.6%)	0.1359 (16.5%)
Matrix Completion	0.0886 (28.1%)	0.1331 (14.7%)

Table 3.7: Performance comparison for missing data estimation for MCAR and MAR settings on the Biobank dataset

We compare the RMSE of M-RNN architecture against four competitive benchmarks: [CPC18], Cubic Interpolation, MissForest, and Matrix Completion in both MCAR and MAR settings. As can be seen in Table 3.7, M-RNN outperforms other state-of-the-art imputation methods in both MCAR and MAR settings.

3.5 Conclusion

The problem of reconstructing/estimating missing data is ubiquitous in many settings – especially in longitudinal medical datasets – and is of enormous importance for many reasons, including statistical analysis, diagnosis, prognosis and treatment. In this chapter we have presented a new method, based on a novel deep learning architecture, for reconstructing/estimating missing data that exploits both the correlation within data streams and the correlation across data streams. We have demonstrated on the basis of a variety of real-world medical datasets that our method makes large and statistically significant improvements in comparison with state-of-the-art benchmarks.

CHAPTER 4

INVASE: Instance-wise Variable Selection using Neural Networks

High-dimensional data is becoming more readily available, and it brings with it a growing need to be able to efficiently select which features to use for a variety of problems. When doing predictions, it is well known that using too many variables with too few samples can lead to overfitting, which can significantly hinder the performance of predictive models. In the realm of interpretability, the large dimensionality of the data is often too much information to present to a human who may be using the machine learning model as a support system. Understanding which features are most relevant to an outcome or to a model output is an important first step in improving predictions and interpretability and many works exist that tackle feature selection on a global level. However, in the heterogeneous data we typically encounter, the prediction made by a model (and indeed the true label) may rely on a different subset of the features for different subgroups within the data [KLA15]. In this chapter we propose a novel *instance-wise* feature selection method, INVASE (INstance-wise VAriable SElection), which attempts to learn which subset of the features is relevant for each sample, allowing us to display the minimal information required to explain each prediction and also to reduce overfitting of predictive models.

Discovering a global subset of relevant features for a particular task is a well-studied problem and there are several existing methods for solving it such as Sequential Correlation Feature Selection [Hal99], Mutual Information Feature Selection [PLD05], Knockoff models [CFJ16], and more [GE03, KR92]. However, global feature selection suffers from a key limitation - the features discovered by global feature selection are the same for all samples. In many cases, in particular when populations are highly heterogeneous, the relevant features

may differ across samples [YZS18b, YZB18]. For instance, different patient subgroups have different relevant features for predicting heart failure [KLA15]. *Instance-wise* feature selection methods such as [CSW18, SGK17] instead try to discover the features that are relevant *for each sample*. When the goal is to provide an interpretable explanation of the predictions made, a key challenge is in ensuring that we do not *over-explain* by providing too much information (i.e. choosing too many features). Naturally, by performing feature selection on an individualized level we are able to select features that are more relevant to each sample, rather than having to choose the top k features globally, which may not explain the predictions for some samples very well, but simply perform well on average across all samples.

In this chapter, we propose a novel instance-wise feature selection method which we term INVASE. We draw influence from actor-critic models [PS08] to solve the problem of backpropagating through subset sampling. Our model consists of 3 neural networks: a selector network, a predictor network and a baseline network. During training, each of these are trained iteratively, with the selector network being trained to minimize a Kullback-Leibler (KL) divergence between the full conditional distribution and the selected-features-only conditional distribution of the outcome. Our model is capable of discovering a different number of relevant variables for each sample which is a key limitation in existing instance-wise approaches (such as [CSW18]). We show significant improvements over the state-of-the-art in both synthetic data and real-word data in terms of true positive rates, false discovery rates, and show better predictive performance with respect to several prediction metrics. Our model can also be easily extended to handle both continuous and discrete outputs and time-series inputs.

4.1 Related works

There are many existing works on global variable selection (see [GE03] for a good summary paper). [PLD05] and [Hal99] use max-dependency min-redundancy criteria [LHL15] with mutual information and Pearson correlation, respectively. [CFJ16] uses multiple hypothesis testing for global variable selection. As noted above, these global selection methods are not

capable of learning sample-specific relevance.

Instance-wise variable selection is also closely related to model interpretation methods. Some previous works are based on backpropagation from the output of the predictive model to the input variables [SVZ13]. DeepLIFT [SGK17] decomposes the output of the neural network on a reference input to compute the contribution of each input variable. However, both methods need white-box access to the pre-trained predictive models to compute the gradient and decomposition. [BSH10] approximates the predictive models using a Parzen window approximator when there is only black-box access to the predictive models. Some other works are based on input perturbation such as [BBM15], [KSM16], [SK14] and [DSZ16]. [LL17] uses Shapley values to compute the variable importance, and [RSG16] uses locally linear models to explain the linear dependency for each sample. [LEL18] tries to interpret tree ensemble models using Shapley values but cannot generalize to other predictive models such as neural networks.

Our work is most closely related to L2X (Learning to Explain) [CSW18]. However, there are 3 key differences between our work and theirs. In L2X, they try to maximize a lower bound of the mutual information between the target Y and the selected input variables X_S. In contrast, we try to minimize the KL divergence between the conditional distributions $Y|X$ and $Y|X_S$. In order to be able to backpropagate through subset sampling, L2X use the Gumbel-softmax trick [JGP16] to approximately discretize the continuous outputs of the neural network. In our work, we use methods from actor-critic models [PS08] to bypass backpropagation through the sampling and instead use the predictor network to provide a reward to the selector network. Finally, due to the Gumbel-softmax used in L2X, the number of variables to be detected must be fixed in advance and is necessarily the same for every sample. The actor-critic methodology used in our model has no such limitations and so we are able to flexibly select a different number of relevant variables for each sample and instead induce sparsity using an l_0 penalty term. In fact, using the actor-critic methodology allows us to directly use the l_0 penalty term (which is not differentiable and therefore not practical to use in general).

4.2 Problem formulation

Let $\mathcal{X} = \mathcal{X}_1 \times ... \times \mathcal{X}_d$ be a d-dimensional feature space and $\mathcal{Y} = \{1, ..., c\}$ be a discrete label space. Let $\mathbf{X} = (X_1, ..., X_d) \in \mathcal{X}$ and $Y \in \mathcal{Y}$ be random variables with joint density (or mass) p and marginal densities (or masses) p_X and p_Y respectively. We will refer to $\mathbf{s} \in \{0, 1\}^d$ as the selection vector, where $s_i = 1$ will indicate that variable i is selected, and $s_i = 0$ will indicate that variable i is not selected. Let $*$ be any point not in any of the spaces $\mathcal{X}_1, ..., \mathcal{X}_d$ and define $\mathcal{X}_i^* = \mathcal{X}_i \cup \{*\}$ and $\mathcal{X}^* = \mathcal{X}_1^* \times ... \times \mathcal{X}_d^*$. Given $\mathbf{x} \in \mathcal{X}$ we will write $\mathbf{x}^{(\mathbf{s})}$ to denote the suppressed feature vector defined by

$$x_i^{(\mathbf{s})} = \begin{cases} x_i \text{ if } s_i = 1 \\ * \text{ if } s_i = 0 \end{cases}$$

so that $*$ represents that a feature is not selected.

In the global feature selection literature, the goal is to find the smallest $\mathbf{s}$ (i.e. the one with fewest 1s) such that $\mathbb{E}(Y|\mathbf{X}^{(\mathbf{s})}) = \mathbb{E}(Y|\mathbf{X})$, or equivalently such that the conditional distribution of Y given $\mathbf{X}^{(\mathbf{s})}$ is the same as Y given all of $\mathbf{X}$. Note that this definition is given fully in terms of random variables, rather than realizations of those random variables.

In contrast, our problem necessarily needs to be defined in terms of realizations since we are aiming to select features *for a given realization*. We will write $\mathbf{x}$ to denote realizations of the random variable $\mathbf{X}$. Then we formalize our problem as one of finding a selector function, $S : \mathcal{X} \to \{0, 1\}^d$ such that for almost every $\mathbf{x} \in \mathcal{X}$ (w.r.t. p_X) we have

$$(Y|\mathbf{X}^{(S(\mathbf{x}))} = \mathbf{x}^{(S(\mathbf{x}))}) \overset{d.}{=} (Y|\mathbf{X} = \mathbf{x}) \tag{4.1}$$

where $\overset{d.}{=}$ denotes equality in distribution and $S(\mathbf{x})$ is minimal (i.e. fewest 1s) such that the constraint (4.1) holds.

We suppose that we have a dataset $\mathcal{D} = \{(\mathbf{x}_j, y_j)\}_{j=1}^n$ consisting of n i.i.d. realizations

of the pair $(\mathbf{X}, Y)$.[1] Note that Y can be viewed as having either come from a dataset, in which case the problem is of selecting predictive features, or as having come from a predictive model, in which case the problem is of explaining the model's predictions.

4.2.1 Optimization problem

In order to learn a suitable selector function, we transform the constraint (4.1) into a soft constraint using the Kullback-Leibler (KL) divergence which, for random variables W and V with densities p_W and p_V is defined as

$$KL(W||V) = \mathbb{E}\left[\log\left(\frac{p_W(W)}{p_V(W)}\right)\right].$$

We define the following loss for our selector function S

$$\mathcal{L}(S) = \mathbb{E}_{\mathbf{x}\sim p_X}\left[KL(Y|\mathbf{X}=\mathbf{x}||Y|\mathbf{X}^{(S(\mathbf{x}))}=\mathbf{x}^{(S(\mathbf{x}))}) + \lambda||S(\mathbf{x})||\right] \tag{4.2}$$

where $||\cdot||$ simply denotes the number of non-zero entries of a vector (or equivalently in this case, the number of 1s) and λ is a hyper-parameter that trades off between the constraint in Equation (4.1) and the number of selected features. The KL divergence in Equation (4.2) can be rewritten as

$$\begin{aligned}
KL(Y|\mathbf{X}=\mathbf{x}||Y|\mathbf{X}^{(S(\mathbf{x}))}=\mathbf{x}^{(S(\mathbf{x}))}) &= \mathbb{E}_{y\sim Y|\mathbf{X}=\mathbf{x}}\left[\log\left(\frac{p_Y(y|\mathbf{x})}{p_Y(y|\mathbf{x}^{(S(\mathbf{x}))})}\right)\right] \\
&= \mathbb{E}_{y\sim Y|\mathbf{X}=\mathbf{x}}\left[\log(p_Y(y|\mathbf{x})) - \log(p_Y(y|\mathbf{x}^{(S(\mathbf{x}))}))\right] \\
&= \int_{\mathcal{Y}} p_Y(y|\mathbf{x})\left[\log(p_Y(y|\mathbf{x})) - \log(p_Y(y|\mathbf{x}^{(S(\mathbf{x}))}))\right] dy
\end{aligned}$$

where $p_Y(\cdot|\cdot)$ denotes the appropriate conditional densities of Y. We will write

$$l(\mathbf{x}, \mathbf{s}) = \int_{\mathcal{Y}} p_Y(y|\mathbf{x})\left[\log(p_Y(y|\mathbf{x})) - \log(p_Y(y|\mathbf{x}^{(s)}))\right] dy \tag{4.3}$$

[1] We will occasionally abuse notation and write y_i to denote the ith element of the one-hot encoding of y, though the context should make it clear when this is the case.

so that our final loss can be written as

$$\mathcal{L}(S) = \mathbb{E}_{\mathbf{x}\sim p_X}\left[l(\mathbf{x}, S(\mathbf{x})) + \lambda||S(\mathbf{x})||\right] \tag{4.4}$$

where $||\cdot||$ denotes the l_0 (pseudo-)norm.

4.3 Proposed model

There are two main challenges in minimizing the loss in Equation (4.4). First, the output space of the selector function ($\{0,1\}^d$) is large - its size increases exponentially with the dimension of the feature space; thus a complete search is impractical in high dimensional settings (and it should be noted that it is in high dimensional settings where feature selection is most necessary). Second, we do not have access to the densities $p_Y(\cdot|\mathbf{x}^{(S(\mathbf{x}))})$ and $p_Y(y|\mathbf{x})$ required to compute Equation (4.4).

4.3.1 Loss estimation

To approximate the densities in Equation (4.3), we introduce a pair of functions $f^\phi : \mathcal{X}^* \times \{0,1\}^d \to [0,1]^c$ parameterized by ϕ and $f^\gamma : \mathcal{X} \to [0,1]^c$ parameterized by γ that will estimate $p_Y(\cdot|\mathbf{x}^{(S(\mathbf{x}))})$ and $p_Y(\cdot|\mathbf{x})$ respectively.

4.3.1.1 Predictor network

We refer to f^ϕ as the predictor network. This will take as input a suppressed[2] feature vector $\mathbf{x}^{(\mathbf{s})}$ and its corresponding selection vector $\mathbf{s}$ and will output a probability distribution (using a softmax layer) over the c-dimensional output space.

f^ϕ is trained to minimize the cross entropy loss given by

$$l_1(\phi) = -\mathbb{E}_{(\mathbf{x},y)\sim p, \mathbf{s}\sim\pi_\theta(\mathbf{x},\cdot)}\Big[\sum_{i=1}^{c} y_i \log(f_i^\phi(\mathbf{x}^{(\mathbf{s})}, \mathbf{s}))\Big]$$

[2]When implemented we set $* = 0$ and include the selection vector to differentiate this from the case $x_i = 0$.

where y_i is the ith component of the one-hot encoding of y and π_θ is the distribution induced by our selector network which will be defined in the following section. f^ϕ is implemented as a fully connected neural network[3].

4.3.1.2 Baseline network

We refer to f^γ as the baseline network, which is standard in the actor-critic literature for variance reduction. f^γ is implemented as a fully connected neural network and is trained to minimize

$$l_3(\gamma) = -\mathbb{E}_{(\mathbf{x},y)\sim p}\Big[\sum_{i=1}^{c} y_i \log(f_i^\gamma(\mathbf{x}))\Big].$$

For fixed ϕ, γ we define our loss estimator, $\hat{l}$, by

$$\hat{l}(\mathbf{x}, \mathbf{s}) = -\left[\sum_{i=1}^{c} y_i \log(f_i^\phi(\mathbf{x}^{(\mathbf{s})}, \mathbf{s})) - \sum_{i=1}^{c} y_i \log(f_i^\gamma(\mathbf{x}))\right]. \tag{4.5}$$

4.3.2 Selector function optimization

We approximate the selector function $S : \mathcal{X} \to \{0,1\}^d$ by using a single neural network, $\hat{S}^\theta : \mathcal{X} \to [0,1]^d$ parameterized by weights θ, that outputs a probability for selecting each feature (i.e. the ith component of $\hat{S}^\theta(\mathbf{x})$ will denote the probability with which we select the ith feature). The selector network induces a probability distribution over the selection space ($\{0,1\}^d$), with the probability of a given joint selection vector $\mathbf{s} \in \{0,1\}^d$ being given by[4]

$$\pi_\theta(\mathbf{x}, \mathbf{s}) = \Pi_{i=1}^{d} \hat{S}_i^\theta(\mathbf{x})^{s_i}(1 - \hat{S}_i^\theta(\mathbf{x}))^{1-s_i}.$$

[3] f^ϕ, f^γ and $\hat{S}^\theta$ could also be implemented as CNNs or RNNs, when appropriate.

[4] Note that, when d is large, this becomes vanishingly small, however, π_θ appears in our loss only via its log and so in practice this is not a problem.

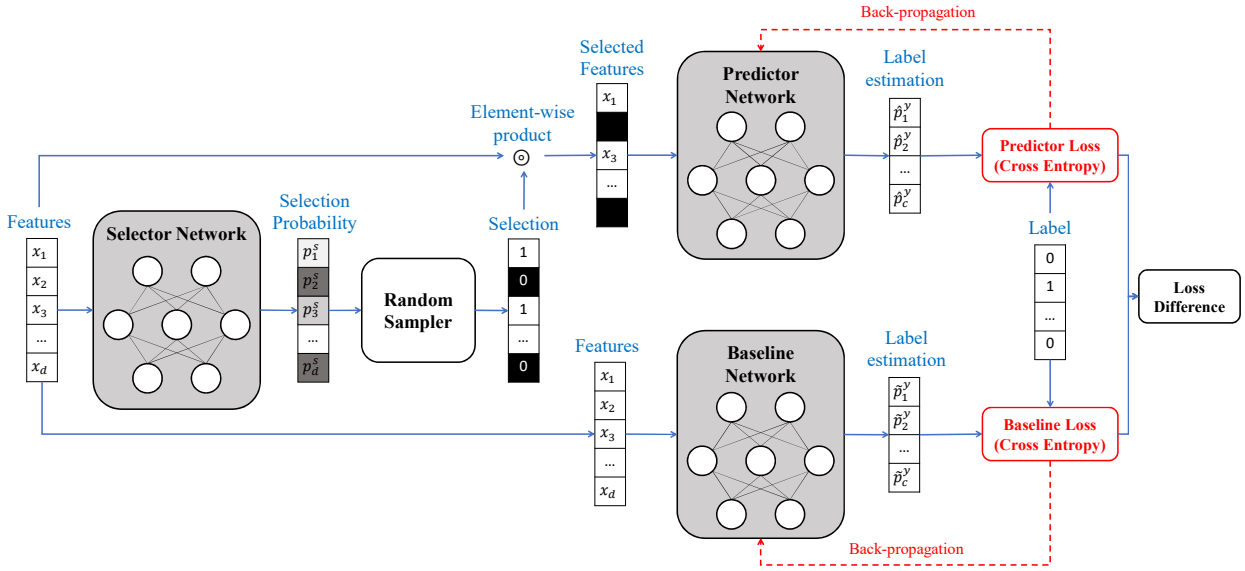

Figure 4.1: Block diagram of INVASE. Instances are fed into the selector network which outputs a vector of selection probabilities. The selection vector is then sampled according to these probabilities. The predictor network then receives the selected features and makes a prediction and the baseline network is given the entire feature vector and makes a prediction. Each of these networks are trained using backpropagation using the real label. The loss of the baseline network is then subtracted from the prediction network's loss and this is used to update the selector network.

Using this, we define the following loss for our selector network

$$\begin{aligned} l_2(\theta) &= \mathbb{E}_{(\mathbf{x},y)\sim p}\left[\mathbb{E}_{\mathbf{s}\sim\pi_\theta(\mathbf{x},\cdot)}\left[\hat{l}(\mathbf{x},\mathbf{s}) + \lambda||\mathbf{s}||_0\right]\right] \\ &= \int_{\mathcal{X}\times\mathcal{Y}} p(\mathbf{x},y)\left(\sum_{\mathbf{s}\in\{0,1\}^d}\pi_\theta(\mathbf{x},\mathbf{s})\left(\hat{l}(\mathbf{x},\mathbf{s}) + \lambda||\mathbf{s}||_0\right)\right)dxdy. \end{aligned}$$

Taking the gradient of this loss with respect to θ gives us

$$\begin{aligned}
\nabla_\theta l_2(\theta) &= \int_{\mathcal{X}\times\mathcal{Y}} p(\mathbf{x}, y) \left(\sum_{\mathbf{s}\in\{0,1\}^d} \nabla_\theta \pi_\theta(\mathbf{x}, \mathbf{s}) \left(\hat{l}(\mathbf{x}, \mathbf{s}) + \lambda ||\mathbf{s}||_0 \right) \right) dxdy \\
&= \int_{\mathcal{X}\times\mathcal{Y}} p(\mathbf{x}, y) \left(\sum_{\mathbf{s}\in\{0,1\}^d} \frac{\nabla_\theta \pi_\theta(\mathbf{x}, \mathbf{s})}{\pi_\theta(\mathbf{x}, \mathbf{s})} \pi_\theta(\mathbf{x}, \mathbf{s}) \left(\hat{l}(\mathbf{x}, \mathbf{s}) + \lambda ||\mathbf{s}||_0 \right) \right) dxdy \\
&= \int_{\mathcal{X}\times\mathcal{Y}} p(\mathbf{x}, y) \left(\sum_{\mathbf{s}\in\{0,1\}^d} \nabla_\theta \log \pi_\theta(\mathbf{x}, \mathbf{s}) \pi_\theta(\mathbf{x}, \mathbf{s}) \left(\hat{l}(\mathbf{x}, \mathbf{s}) + \lambda ||\mathbf{s}||_0 \right) \right) dxdy \\
&= \mathbb{E}_{(\mathbf{x},y)\sim p} \left[\mathbb{E}_{\mathbf{s}\sim\pi_\theta(\mathbf{x},\cdot)} \left[\left(\hat{l}(\mathbf{x}, \mathbf{s}) + \lambda ||\mathbf{s}||_0 \right) \nabla_\theta \log \pi_\theta(\mathbf{x}, \mathbf{s}) \right] \right].
\end{aligned}$$

We update each of $\hat{S}^\theta$, f^ϕ and f^γ iteratively using stochastic gradient descent. Pseudo-code of INVASE is given in Algorithm 2 and a block representation of INVASE can be found in Fig. 4.1.

4.4 Experiments

In this section, we quantitatively evaluate INVASE against various state-of-the-art benchmarks on both synthetic and real-world datasets. We evaluate our performance both at identifying ground truth relevance and at enhancing predictions. We compare our model with 4 global variable selection models: Knockoffs [CFJ16], Tree Ensembles (Tree) [GEW06], Sequential Correlation Feature Selection (SCFS) [Hal99], and LASSO regularized linear model; and 3 instance-wise feature selection methods: L2X [CSW18], LIME [RSG16], and Shapley [LL17]. Implementation of INVASE can be found at `https://github.com/jsyoon0823/INVASE`.

Algorithm 2 Pseudo-code of INVASE

Inputs: learning rates $\alpha, \beta > 0$, mini-batch size $n_{mb} > 0$, dataset $\mathcal{D}$

Initialize parameters θ, ϕ, γ

while Converge **do**

Sample a mini-batch from the dataset $(\mathbf{x}_j, y_j)_{j=1}^{n_{mb}} \sim \mathcal{D}$

for $j = 1, ..., n_{mb}$ **do**

Calculate selection probabilities

$$(p_1^j, ..., p_d^j) \leftarrow \hat{S}^\theta(\mathbf{x}_j)$$

Sample selection vector

for $i = 1, ..., d$ **do**

$$s_i^j \sim Ber(p_i^j)$$

Calculate loss

$$\hat{l}_j(\mathbf{x}_j, \mathbf{s}_j) \leftarrow -\left[\sum_{i=1}^{c} y_i^j \log(f_i^\phi(\mathbf{x}_j^{(\mathbf{s}_j)}, \mathbf{s}_j)) - \sum_{i=1}^{c} y_i^j \log(f_i^\gamma(\mathbf{x}_j))\right]$$

Update the selector network parameters θ

$$\theta \leftarrow \theta - \alpha \frac{1}{n_{mb}} \sum_{j=1}^{n_{mb}} \left(\hat{l}_j(\mathbf{x}_j, \mathbf{s}_j) + \lambda ||\mathbf{s}_j||\right) \nabla_\theta \log \pi_\theta(\mathbf{x}_j, \mathbf{s}_j)$$

Update the predictor network parameters ϕ

$$\phi \leftarrow \phi - \beta \frac{1}{n_{mb}} \sum_{j=1}^{n_{mb}} \sum_{i=1}^{c} y_i^j \times \nabla_\phi \log(f_i^\phi(\mathbf{x}_j^{(\mathbf{s}_j)}, \mathbf{s}_j))$$

Update the baseline network parameters γ

$$\gamma \leftarrow \gamma - \beta \frac{1}{n_{mb}} \sum_{j=1}^{n_{mb}} \sum_{i=1}^{c} y_i^j \times \nabla_\gamma \log(f_i^\gamma(\mathbf{x}_j))$$

4.4.1 Synthetic data experiments

4.4.1.1 Experimental settings

For our first set of experiments, we use the same synthetic data generation models as in L2X [CSW18]. The input features are generated from an 11-dimensional [5] Gaussian distribution with no correlations across the features ($\mathbf{X} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$). The label Y is sampled as a Bernoulli random variable with $\mathbb{P}(Y = 1|\mathbf{X}) = \frac{1}{1+\text{logit}(\mathbf{X})}$, where $\text{logit}(\mathbf{X})$ is varied to create 3 different synthetic datasets:

- **Syn1:** $\exp(X_1 X_2)$
- **Syn2:** $\exp(\sum_{i=3}^{6} X_i^2 - 4)$
- **Syn3:** $-10 \times \sin 2X_7 + 2|X_8| + X_9 + \exp(-X_{10})$

In each of these datasets, the label depends on the same subset of features for every sample. To highlight the capability of INVASE to detect instance-wise dependence, we generate 3 further synthetic datasets as follows:

- **Syn4:** If $X_{11} < 0$, logit follows **Syn1**, otherwise, logit follows **Syn2**.
- **Syn5:** If $X_{11} < 0$, logit follows **Syn1**, otherwise, logit follows **Syn3**.
- **Syn6:** If $X_{11} < 0$, logit follows **Syn2**, otherwise, logit follows **Syn3**.

Note that in **Syn4** and **Syn5**, the number of relevant features is different for different samples.

For each of **Syn1** to **Syn6** we draw 20,000 samples from the data generation model and separate each into training ($\mathcal{D}_{train} = (\mathbf{x}_i, y_i)_{i=1}^{10000}$) and testing ($\mathcal{D}_{test} = (\mathbf{x}_j, y_j)_{j=1}^{10000}$) sets. For each method we try to find the top k relevant features for each sample (we set $k = 4$ for **Syn1, Syn2, Syn3, Syn4, Syn5** and $k = 5$ for **Syn6**), note, however, that k is not given as an input to INVASE (but is necessary for other methods). The performance metrics we use are the true positive rate (TPR) (higher is better) and false discovery rate (FDR)

[5] We also perform experiments using 100 features to demonstrate the scalability of our method.

(lower is better) to measure the performance of the methods when the focus is on discovery (i.e. discovering which features are relevant) and we use Area Under the Receiver Operating Characteristic Curve (AUROC) and Area Under the Precision Recall Curve (AUPRC) when the focus is on predictions.

4.4.1.2 Discovery

Dataset	**Syn1**		**Syn2**		**Syn3**		**Syn4**		**Syn5**		**Syn6**	
Metrics (%)	TPR	FDR	TPR	FDR	TPR	FDR	TPR	FDR	TPR	FDR	TPR	FDR
INVASE	**100.0**	**0.0**	**100.0**	**0.0**	**92.0**	**0.0**	**99.8**	**10.3**	**84.8**	**1.1**	**90.1**	**7.4**
L2X	100.0	0.0	100.0	0.0	69.4	30.6	79.5	21.8	74.8	26.3	83.3	16.7
LIME	13.8	86.2	100.0	0.0	98.1	1.9	40.7	49.4	41.1	50.6	50.5	49.5
Shapley	60.4	39.6	93.3	6.7	90.9	9.1	65.2	31.9	62.9	33.7	71.2	28.8
Knockoff	10.0	70.0	8.7	36.2	81.2	17.5	38.8	35.1	41.0	51.1	56.6	42.1
Tree	100.0	0.0	100.0	0.0	100.0	0.0	54.7	39.0	56.8	37.5	60.0	40.0
SCFS	23.5	76.5	39.5	60.5	78.3	22.0	48.9	52.4	42.4	51.2	56.1	43.9
LASSO	19.0	81.0	39.8	60.2	78.3	21.7	49.9	50.9	45.5	48.2	56.4	43.6

Table 4.1: Relevant feature discovery results for Synthetic datasets with 11 features

As demonstrated by Table 4.1, our method is capable of detecting relevant features on a global level (**Syn1**, **Syn2** and **Syn3**) as well as on an instance-wise level (**Syn4**, **Syn5** and **Syn6**) outperforming all other methods in both cases (both global and instance-wise methods). The particularly poor performance of some global feature selection methods in **Syn1**, **Syn2** and **Syn3** (where there is no instance-wise relevance) is due to the non-linearity of the relationship between features and labels.

The results for **Syn4**, **Syn5** and **Syn6** demonstrate that INVASE is capable of detecting a different number of relevant features for each sample when necessary - the performance improvement over L2X is greater in Syn4 and Syn5 than Syn6. In particular, in Syn4, L2X is forced to overselect features when $X_{11} < 0$ and underselect when $X_{11} \geq 0$ thus resulting in higher FDR and lower TPR, respectively. To highlight this, in Table 4.2 we report the group specific FDR and TPR on Syn4 and Syn5 when setting $k = 3, 4, 5$, where Group 1 refers to samples with $X_{11} < 0$ and Group 2 to samples with $X_{11} \geq 0$.

For $k = 3$ in Syn4, we see that INVASE and L2X have comparable FDR in Group 1,

Datasets	**Syn4**				**Syn5**			
Group	1		2		1		2	
Metrics (%)	TPR	FDR	TPR	FDR	TPR	FDR	TPR	FDR
INVASE	99.5	24.6	100.0	0.4	69.2	1.6	99.8	0.6
L2X ($k = 3$)	71.1	28.9	57.2	4.6	65.5	34.5	55.4	7.7
L2X ($k = 4$)	81.0	39.2	74.9	6.3	76.2	42.9	72.4	9.4
L2X ($k = 5$)	89.9	46.0	84.6	15.4	87.5	47.5	82.1	17.9

Table 4.2: Detailed comparison of INVASE with L2X in Syn4 and **Syn5**, highlighting the capability of INVASE to select a flexible number of features for each sample. Group 1: $X_{11} < 0$, Group 2: $X_{11} \geq 0$

since the total number of relevant features for each sample is 3 (X_1, X_2, X_{11}). However, when we increase k, we see that the FDR increases for L2X as it is forced to select more than 3 features, which necessarily means that the FDR must be *at least* 40% even if L2X was finding the relevant features perfectly. On the other hand, for Group 2 we see that the TPR is low for $k = 3$ since necessarily, L2X cannot possibly select all of the 5 relevant features. INVASE, however, is able to select the correct number in both and hence enjoys low FDR and high TPR.

Syn5 reinforces the conclusions we drew for L2X in Syn4. Interestingly, though, for INVASE, we found that X_{11} was almost never selected for Group 1 in Syn5. We believe this is because the lack of overlap between the relevant features for each group means that the predictor network can essentially learn two separate networks - one for each group. This is because it is possible to create two subnetworks with non-overlapping weights that each take as input the features of a given group. X_{11} is therefore unnecessary for prediction. Note, however, that X_{11} is *highly relevant* for the selector network in deciding which features to pass on and so it is not true that X_{11} isn't relevant, but simply that the selector network does not need to "pass on" its relevance to the predictor network.

4.4.1.3 High dimensional discovery

To demonstrate the scalability of our method, we run an experiment in which we increase the total number of features to 100. The features are generated as a 100-dimensional Gaussian with no correlations ($\mathcal{N}(\mathbf{0}, \mathbf{I})$) and the relationships between features and label remains as in Table 4.1 (i.e. we are adding 89 additional noisy signals that have no effect on the label).

Dataset	**Syn1**		**Syn2**		**Syn3**		**Syn4**		**Syn5**		**Syn6**	
Metrics (%)	TPR	FDR	TPR	FDR	TPR	FDR	TPR	FDR	TPR	FDR	TPR	FDR
INVASE	**100.0**	**0.0**	**100.0**	**0.0**	**100.0**	**0.0**	**66.3**	**40.5**	**73.2**	**23.7**	**90.5**	**15.4**
L2X	6.1	93.9	81.4	18.6	57.7	42.3	48.5	46.4	35.4	60.8	66.3	33.7
LIME	0.0	100.0	**100.0**	**0.0**	92.7	7.3	43.8	47.4	42.3	50.1	50.1	49.9
Shapley	4.4	95.6	95.1	4.9	88.8	11.2	50.2	43.4	49.9	44.2	62.5	37.5
Knock off	0.0	64.9	3.7	71.2	74.9	24.9	28.2	59.8	33.1	59.4	46.9	53.0
Tree	49.9	50.1	100.0	0.0	100.0	0.0	40.7	49.5	56.7	37.5	58.4	41.6
SCFS	2.5	97.5	5.3	94.7	74.9	25.1	27.0	74.6	30.6	62.1	38.3	61.7
LASSO	2.5	97.5	4.0	96.0	75.3	24.7	28.3	73.2	36.0	56.9	45.9	54.1

Table 4.3: Relevant feature discovery for synthetic datasets with 100 features

As can be seen in Table 4.3, INVASE also works consistently better than all other benchmarks in all 6 synthetic datasets in this setting. In fact, we see a significant reduction in performance (compared to the 11 feature setting) for L2X in Syn1, with the TPR dropping more than 90% leading to an almost complete failure of the method to detect any relevant features. In particular, we see that L2X does not scale as well as INVASE with the dimensionality of the data, which is particularly limiting for a feature selection method.

4.4.1.4 Prediction

In this experiment we analyze the effect of using feature selection as a pre-processing step for prediction. We first perform feature selection (either instance-wise or global) and then train a 3-layer fully connected network with Batch Normalization [IS15] in every layer (to avoid overfitting) to perform predictions on top of the (feature-selected) data. In this setting we compare the two global feature selection methods (LASSO and Tree) and one instance-wise feature selection method (L2X). Furthermore, we also compare with the predictive

model without any feature selections (w/o FS) and the predictive model with ground truth globally relevant features[6] (with Global). In particular, this allows us to demonstrate that the improvements in prediction performance are not just because the global feature selection performed implicitly by INVASE is better than the other global feature selection methods but are also due to the fact that we select features on an instance-wise level. Experiments here are conducted on synthetic data with 100 features but the same labelling procedures as above.

Dataset	**AUROC**					
	w/o FS	with Global	**with INVASE**	with Tree	with L2X	with LASSO
Syn1	.578±.004	.686±.005	.690±.006	.574±.101	.498±.005	.498±.006
Syn2	.789±.003	.873±.003	.877±.003	.872±.003	.823±.029	.555±.061
Syn3	.854±.004	.900±.003	.902±.003	.899±.001	.862±.009	.886±.003
Syn4	.558±.021	.774±.006	.787±.004	.684±.017	.678±.024	.514±.031
Syn5	.662±.013	.784±.005	.784±.005	.741±.004	.709±.008	.691±.024
Syn6	.692±.015	.858±.004	.877±.003	.771±.031	.827±.017	.727±.025
Dataset	**AUPRC**					
	w/o FS	with Global	**with INVASE**	with Tree	with L2X	with LASSO
Syn1	.567±.007	.690±.006	.694±.006	.577±.102	.498±.007	.499±.008
Syn2	.799±.005	.878±.005	.886±.004	.878±.004	.817±.031	.591±.037
Syn3	.861±.003	.905±.002	.907±.003	.904±.002	.860±.012	.890±.002
Syn4	.572±.019	.794±.006	.804±.004	.681±.031	.672±.025	.536±.025
Syn5	.665±.019	.796±.005	.797±.006	.765±.003	.719±.011	.680±.040
Syn6	.709±.018	.870±.005	.886±.004	.779±.027	.835±.017	.757±.036

Table 4.4: Prediction performance comparison with and without feature selection methods (L2X, LASSO, Tree, INVASE, and Global). Global is using ground-truth globally relevant features for each dataset

As can be seen in Table 4.4, there is a significant performance improvement when discarding all of the irrelevant features (*with Global*). However, neither of the global feature selection methods (*Tree* and *Lasso*) are capable of achieving this improvement. On the other hand, INVASE is capable of achieving (and beating - in Syn4 and Syn6) this improvement, demonstrating its capability both at selecting features globally better than existing methods

[6]For example, in Syn1 the predictor network in the *with Global* setting is trained on only X_1 and X_2 and in Syn4 it would be trained on $X_1, X_2, X_3, X_4, X_5, X_6, X_{11}$.

but also at improving on global selection with instance-wise selection (where relevant), to provide further improvements. On the other hand, *L2X* performs worse than the global methods in Syn1-3, demonstrating an inability to perform even global feature selection in this higher dimensional setting (this is supported by the high dimensional discovery results in the previous subsection), and in Syn4-6 is performing worse than *with Global* (which now is not even optimal).

Furthermore, even though we include Batch Normalization to avoid overfitting, with a small number of samples and high number of dimensions, the 3-layer fully connected network still suffers from overfitting as demonstrated by the significant difference in performance between *w/o FS* and *with Global*. This demonstrates the necessity of feature selection as a pre-processing step. Lastly, in comparison to *with Global*, with INVASE achieves performance gains in Syn4 and Syn6. It quantitatively shows that instance-wise feature selection can further improves the predictive model from ground truth global feature selection.

4.4.2 Real-world data experiments

4.4.2.1 Data description

In this section we use two real-world datasets to perform a series of further experiments. The first, the Meta-Analysis Global Group in Chronic Heart Failure (MAGGIC) dataset [PAM12], has 40,409 patients each with 31 measured features. The label is all-cause mortality. The second, the Prostate, Lung, Colorectal and Ovarian (PLCO) Cancer Screening Trial in the US and the European Randomized Study of Screening for Prostate Cancer (ERSPC) dataset [GPH00, SHR09] contains 38,001 each with 106 measured features. The label in this dataset is mortality due to prostate cancer. We refer to this as the PLCO dataset.

4.4.2.2 The discovered feature importance in MAGGIC dataset

In this next experiment, we visualize the ability of INVASE to select features on an individualized level. Fig. 4.2(left) shows the selection probability (given by INVASE) of each

feature for 20 randomly selected patients in the MAGGIC dataset. Fig. 4.2(right) shows the selection probability of each feature averaged over different binary splits of the data (i.e. when split into Male and Female). In Table 4.5, we also report the mean and variance of the number of selected features in each subgroup.

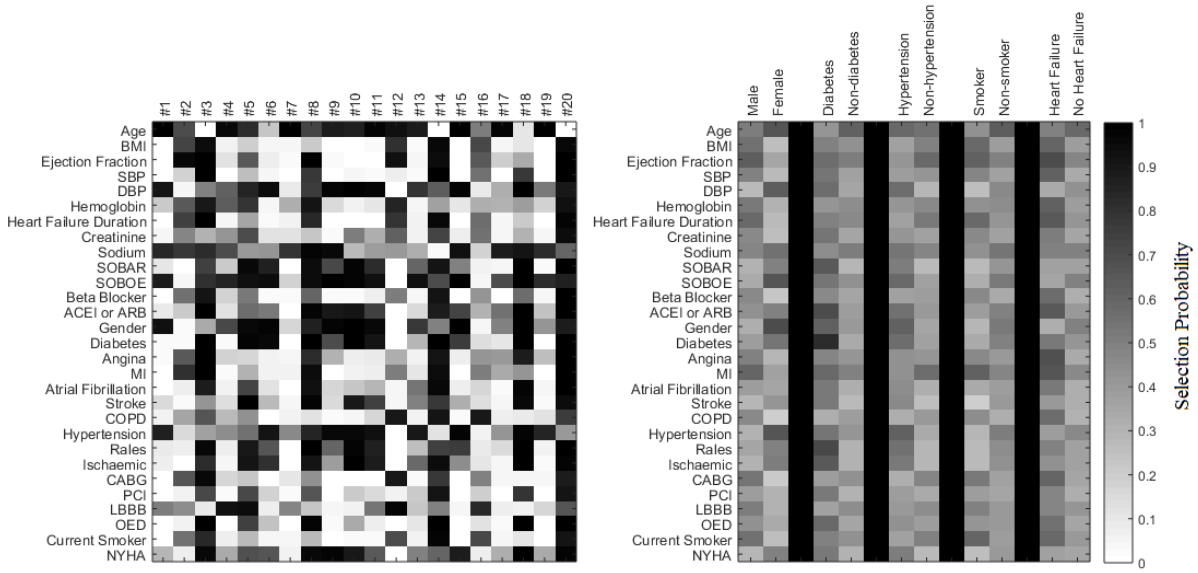

Figure 4.2: Left: The feature importance for each of 20 randomly selected patients in the MAGGIC dataset. Right: The average feature importance for different binary splits in the MAGGIC dataset.

Overall	Male	Diabetes	Hypertension	Smoker	Heart Failure
	43.5±10.7	53.2±10.8	46.6±9.3	41.0±12.1	51.8±11.1
42.5±18.4	Female	Non-diabetes	Non-hypertension	Non-smoker	No Heart Failure
	40.8±15.6	39.3±8.0	40.0±9.3	43.2±7.0	39.6±6.9

Table 4.5: Selection probability of overall and patient subgroups by INVASE in MAGGIC dataset. (Mean ± Std)

As can be seen, INVASE discovers significantly different features for both individuals and for different subgroups of the dataset.

4.4.2.3 Results: Prediction using real data variables with real label

Evaluating the performance of feature selection methods on real data is difficult, since ground truth relevance is often not known. We therefore cannot use TPR and FDR to evaluate the performance on real data. In our final experiment, therefore, we instead focus on prediction performance exactly as in Section 4.4.1.4 (except now both the features and label come from real data).

Datasets	Metrics	AUROC	AUPRC	AUROC	AUPRC
	Labels	**3 year**		**5 year**	
MAGGIC	INVASE	**.722±.005**	**.655±.010**	**.740±.005**	**.867±.006**
	Without INVASE	.720±.006	.639±.009	.730±.006	.855±.004
	Labels	**5 year**		**10 year**	
PLCO	INVASE	**.637±.007**	**.329±.013**	**.673±.007**	**.506±.006**
	Without INVASE	.629±.008	.324±.011	.657±.006	.485±.008

Table 4.6: Prediction performance for MAGGIC and PLCO dataset.

As can be seen in Table 4.6, INVASE consistently improves prediction performance in each of the two settings (different time horizons) in each dataset.

4.4.2.4 Predictive performance comparison on real-world datasets

In this experiment, we evaluate the predictive performance gains of using each feature selection method as a pre-processing step on the two real datasets, MAGGIC and PLCO (as was done for synthetic data in Section 4.4.1.4). For each method, we first perform feature selection and then train a predictive model on top of the feature-selected data, where the model has the same architecture as the INVASE predictor network (to create a fair comparison of methods). As can be seen in Table 4.7, INVASE significantly outperform the other approaches.

Datasets	MAGGIC				PLCO			
Labels	3-year		5-year		5-year		10-year	
Metrics	AUROC	AUPRC	AUROC	AUPRC	AUROC	AUPRC	AUROC	AUPRC
INVASE	0.722	0.655	0.740	0.867	0.637	0.329	0.673	0.506
L2X	0.609	0.529	0.607	0.794	0.558	0.170	0.583	0.365
LIME	0.637	0.5596	0.634	0.808	0.597	0.183	0.601	0.374
Shapley	0.641	0.557	0.617	0.797	0.614	0.194	0.615	0.381
Knockoff	0.686	0.614	0.711	0.853	0.619	0.230	0.658	0.475
Tree	0.678	0.604	0.708	0.850	0.632	0.269	0.655	0.469
SCFS	0.683	0.623	0.723	0.857	0.632	0.231	0.632	0.444
LASSO	0.692	0.615	0.709	0.847	0.623	0.218	0.656	0.467

Table 4.7: Predictive Performance Comparison on two real-world datasets (MAGGIC and PLCO) in terms of AUROC and AUPRC

4.5 Conclusion

In this chapter, we propose a novel instance-wise feature selection method using the actor-critic methodology, which we term INVASE. We demonstrate through a mixture of synthetic and real data experiments that INVASE significantly outperforms state-of-the-art benchmarks.

CHAPTER 5

ASAC: Active Sensing using Actor-Critic models

In many medical settings, making observations is costly [WRG96]. For example, performing lab tests on a patient incurs a cost, both financially as well as causing fatigue to the patient [KBR09, KNS08]. In such settings, the decision to observe is important. This decision involves a trade-off between the value of the information obtained from the observation and the cost of making the observation. This problem presents itself when the data can be observed sequentially, so that we can observe a particular measurement before deciding which other measurements to observe. This problem presents itself in both static and in time-series settings, with the key difference being that in the time-series setting, the values for a given stream[1] will change over time and thus we may wish to re-measure this, whereas in the static setting we know that once we observe a stream, we know its (fixed) value.

Genetic tests, for example, will have the same outcome whether we perform them now or later. As such, it may be advantageous to perform some tests, observe the results, and then decide on further tests to perform based on the results of the first [BKD92]. Because the outcome of the tests will not change over time (we are in a *static setting*), there is no need to perform the tests we have already performed again and also no "worry" that we might miss something by not measuring it now (we can always go back and measure it later).

On the other hand, in an Intensive Care Unit (ICU) setting [EAO07] where important lab tests are being repeated and the results are always changing, we can no longer ignore a stream once it has been measured (its value may have changed since the last time we measured it) and moreover if we decide *not* to measure something, then we have missed our

[1]We use the term stream to refer to both the sequential values of a time-series variable and the single value of a static variable interchangeably.

chance to measure it in that particular instant (we cannot go back and measure its value in the past). We can, however, still use past observations in determining what to measure next.

We refer to the problem of deciding what to observe in the future based on the measurements observed so far as *active sensing* [YKR09, AS16]. This problem presents itself in many healthcare applications [AS16, SWR10]. We formalize the problem of active sensing as a sequential decision making process in which, at each step, we select variables to measure based on all previously selected variables. When selecting variables, we wish to select those which are most predictive of the label, while also minimising cost.

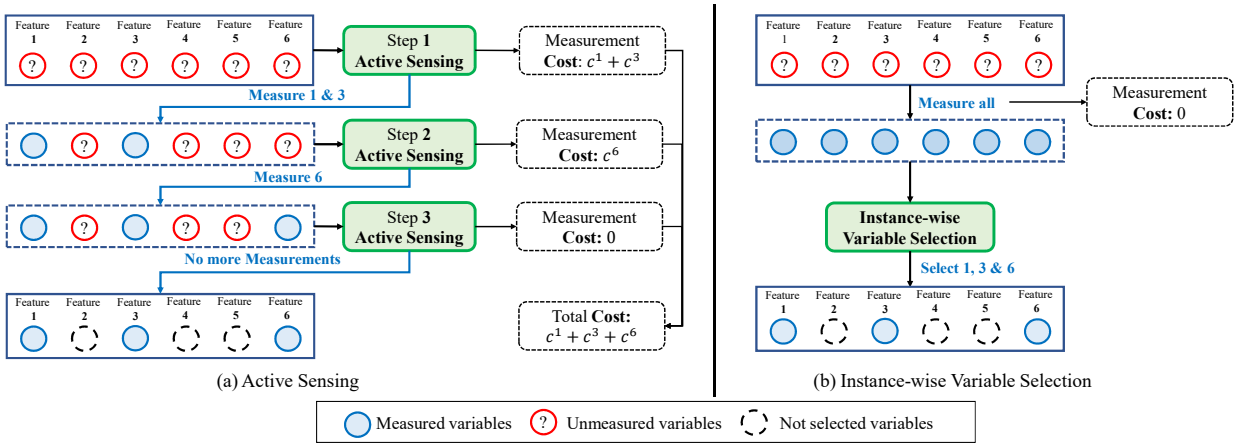

Figure 5.1: Comparison of active sensing and instance-wise variable selection in the static setting.

This formulation of active sensing is related to instance-wise variable selection frameworks such as [YJS19a, CSW18]. In instance-wise variable selection, the goal is to find a minimal subset of variables such that the conditional label distribution is preserved. However, in instance-wise variable selection, all of the variables should be measured *before* making the decision of which to select. In such settings, the goal is to efficiently *summarize* the information present in the entire feature vector in a lower dimensional feature vector. This is typically because the costly part there is not in observing the value of a variable but rather in presenting the value of the variable. In the active sensing framework, the cost has been shifted from presenting the information to measuring it and as such features that are not selected are not measured. Moreover, in the static setting of instance-wise variable selection, only a single selection is made, whereas for active sensing, both in the static and time-series settings, a

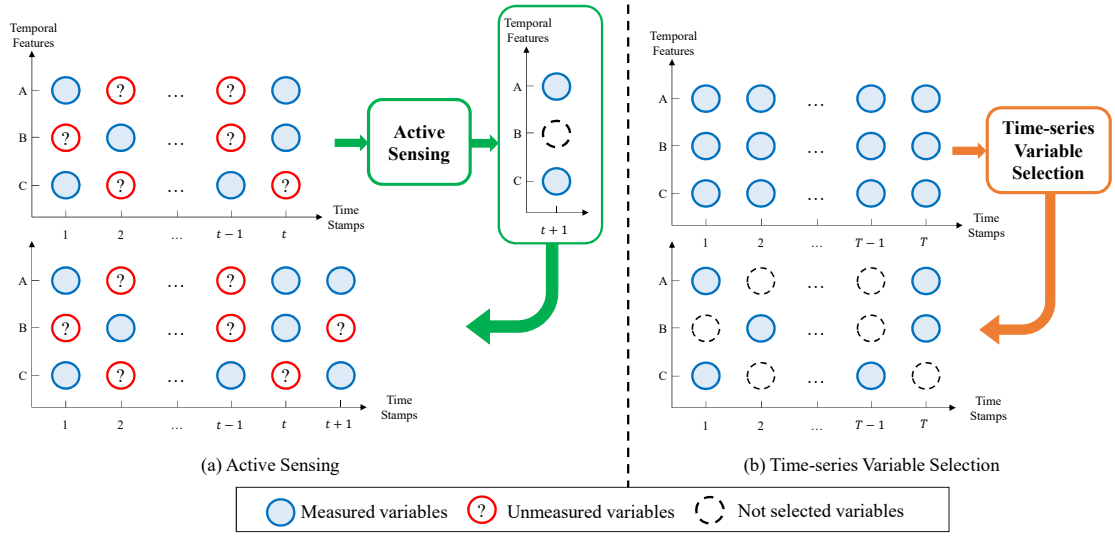

Figure 5.2: Comparison of active sensing and instance-wise variable selection in the time-series setting.

sequence of selections is made. Fig. 5.1 and 5.2 illustrates these differences between active sensing and instance-wise variable selection in static and time-series settings.

In this chapter, we propose ASAC (Active Sensing using Actor-Critic models), an algorithm capable of addressing active sensing in both static and time-series settings. ASAC consists of two networks: a selector network and a predictor network. The selector network uses previously selected features to determine which streams to observe next. The predictor network uses the selected features to predict a label. The networks are trained to minimize a Kullback-Leibler divergence between the conditional label distribution given all features and the conditional label distribution given only the selected features (thus ensuring that the selected features are as predictive of the label as all the features). Cost is introduced by adding a penalty term to the loss. We draw on actor-critic methodology [KT00] to allow "back-propagation" through the sampling process of the selector network. We model each network using LSTMs [HS97] to deal with sequential inputs and outputs, though any sequential model (e.g. temporal convolutions [ODZ16]) could be used.

In our experiments, we demonstrate the efficacy of ASAC in a variety of scenarios using synthetic data. Then, using two real-world medical datasets (ADNI [PAB10] and MIMIC-III [JPS16]) we show that ASAC significantly outperforms the existing state-of-the-art methods.

5.1 Related works

This chapter draws motivation from existing instance-wise variable selection frameworks such as L2X [CSW18], LIME [RSG16], Shapley [LL17], DeepLIFT [SGK17] and in particular, Instance-wise variable selection (INVASE) [YJS19a]. As noted above, a key difference between instance-wise variable selection and active sensing is in *what* is measured *before* making the selection. In addition, each of these works formalize the problem only in the static setting (where there are no temporal features). The applications for this problem are restricted to model interpretation and the models cannot be extended to the active sensing framework.

Deep Sensing [YZS18a] is the work most closely related to ours. Like ASAC, they attempt to solve the active sensing problem using deep learning, especially RNNs. The Deep Sensing framework involves learning 3 different networks: an interpolation network, a prediction network and an error estimation network. Each network is separately optimized for its own objective and then combined together after training to be used for active sensing. On the other hand, ASAC jointly optimizes the selector and predictor networks, *both* for the objective of active sensing, doing so by leveraging ideas from actor-critic methods [KT00]. Furthermore, Deep Sensing treats each feature independently, deciding what to measure by looking at the affect of a single feature on the label *in isolation*. ASAC, on the other hand, jointly estimates the effect of multiple features on the label prediction. This is critical when the features are highly correlated and also when the cost of measuring one feature differs significantly from measuring another noisier correlated feature. In the experiments, we show that our framework significantly outperforms the state-of-the-art in all settings.

Parallels can be drawn between active sensing and attention mechanisms [BCB14, VSP17], though like instance-wise variable selection, attention observes the entire set of measurements and then decides which time points to "focus on". In contrast to instance-wise variable selection, attention is typically applied over time. Attention was first introduced and has been more thoroughly explored as "soft" attention [BCB14, VSP17] in which different time points are weighted (and *not* hard-selected) according to their importance. Active sensing, on the other hand, is only meaningful in a hard-selection setting (since weighting measurements

still requires them to be measured and therefore the cost is still incurred). Hard attention mechanisms do exist [XBK15], but they share the characteristic found in both instance-wise variable selection and soft attention in that all values must be observed before a selection is made.

In [YKR09], they propose a solution for active sensing using a Bayesian approach with Gaussian processes. Data stream are modelled as Gaussian processes and therefore, the complexity of the algorithm increases quadratically in the dimensionality of the data and estimation accuracy decreases quickly with the number of dimensions. [AS16] discuss the active sensing problem for a single data stream observed over time, reducing the problem from *what* and *when* to observe to just *when* to observe. In [AS16], they explicitly model the stream as a given stochastic process and use the characteristics of the assumed process to learn optimal sampling times. This work cannot be applied in the multi-stream setting we investigate in this chapter.

The information bottleneck [TPB00] attempts to find a representation $\tilde{X}$ that is a function of X trading off between maximizing the mutual information between $\tilde{X}$ and Y and minimizing the mutual information between $\tilde{X}$ and X. The first contrast with the active sensing framework is that when constructing $\tilde{X}$, the entire features are used, whereas in active sensing, the decision of what to select is only based on previously selected features. Moreover, $\tilde{X}$ does not necessarily correspond to a subset of the features, but can lie in an entirely different representation space. In ASAC, the selected features are necessarily a subset of the features, and cannot just be any arbitrary mapping of them. We also aim to minimize the cost of the selection, which is not the same as minimizing the mutual information between $\tilde{X}$ and X. Table 5.1 summarizes the comparison of related works.

5.2 Problem formulation

In this section, we first describe the active sensing problem in the static setting, and then explain the differences in the time-series setting.

	Active Sensing	Time-series	Multi-variate	(Non-)Causal	Optimization
ASAC	✓	✓	✓	Causal	Joint
[YZS18a]	✓	✓	✓	Causal	Individual
[QSC17]		✓	✓	Non-causal	Joint
[BCB14]		✓	✓	Non-causal	Joint
[VSP17]		✓	✓	Non-causal	Joint
[YJS19a]			✓	Non-causal	Joint
[AS16]	✓	✓		Causal	Joint

Table 5.1: Comparison of related works. Causal refers to whether or not a selection depends on future selections or not.

5.2.1 Static setting

Let $\mathcal{X} = \mathcal{X}^1 \times ... \times \mathcal{X}^d$ be a d-dimensional feature space and $\mathcal{Y}$ be a label space (either $\mathbb{R}$ for regression problems or $\{1, 2, ..., C\}$ for multi-class classification problems with C classes). We consider random variables $\mathbf{X} \in \mathcal{X}$, $Y \in \mathcal{Y}$ with some joint distribution p (and marginal distributions p_X and p_Y). For each feature, we assume that there is some cost, c^i, where $i = 1, ..., d$, associated with measuring the i-th feature. The cost vector is denoted as $\mathbf{c} = (c^1, ..., c^d)$.

A sensing decision is a vector $\mathbf{s} = (s^1, ..., s^d) \in \{0, 1\}^d$ where $s^i = 1$ corresponds to observing i-th feature. Let $*$ be any point not in $\mathcal{X}^1, ..., \mathcal{X}^d$. For any sensing vector $\mathbf{s}$ and any feature vector $\mathbf{x} = (x^1, ..., x^d) \in \mathcal{X}$ let $\mathbf{x}(\mathbf{s})$ be the vector obtained by

$$x(\mathbf{s})^i = \begin{cases} x^i & \text{if } s^i = 1 \\ * & \text{if } s^i = 0 \end{cases} \tag{5.1}$$

We refer to $\mathbf{x}(\mathbf{s})$ as the observed feature vector. In the static setting, we define a sensing decision sequence as $(\mathbf{s}_1, ..., \mathbf{s}_m)$ where each $\mathbf{s}_j$ sensing decision and we require that if $s^i_{j-1} = 1$, then $s^i_j = 1$ so that the sensing decisions form a nested sequence (this is simply so that $\mathbf{s}_j$ describes fully which features have already been measured at step j) and each $\mathbf{s}_j = \mathbf{s}_j(\mathbf{x}(\mathbf{s}_{j-1}))$ is allowed to depend on $\mathbf{x}(\mathbf{s}_{j-1})$.

Our goal, then, is to find a sensing decision sequence $(\mathbf{s}_1, ..., \mathbf{s}_m)$ that minimizes the total cost of measuring the chosen variables (i.e. $\mathbf{c}^T \mathbf{s}_m = \sum_{i=1}^d c^i \times s^i_m$) subject to the conditional

distribution of Y given $\mathbf{X}$ being equal to the conditional distribution of Y given $\mathbf{X}(\mathbf{s}_m)$. That is, we wish to select variables that still allow us to predict Y as well as if we had measured everything, and among the sets of variables that do this, we wish to find the set with minimal measuring cost.

5.2.2 Time-series setting

In the time-series setting, a few modifications need to be made to the problem formulation given in Section 5.2.1. Instead of considering simple random variables, we now consider an indexed family (or sequence) of these random variables $\mathbf{X} = (\mathbf{X}_t)_{t\in\mathcal{T}}$ and $(Y_t)_{t\in\mathcal{T}}$ where t is an index in some time indexing set $\mathcal{T}$ with $\mathcal{T}$ being some bounded subset of either $\mathbb{R}$ or $\mathbb{N}$. In our case we focus on the discrete setting where $\mathcal{T} = \{1, ..., T\} \subset \mathbb{N}$ where T is some *random* stopping time (whose distribution we absorb into p), with our random processes assumed to be regularly sampled.

In contrast to the static setting, a sensing decision sequence now no longer requires that if $s^i_{t-1} = 1$, then $s^i_t = 1$, since now the values for each component of the process may vary between decisions and so will need to be remeasured if selected again (thus incurring a new cost). In addition, each sensing decision is allowed to depend on *all* observations made so far, that is $\mathbf{s}_t = \mathbf{s}_t(\mathbf{x}(\mathbf{s}_1), ..., \mathbf{x}(\mathbf{s}_{t-1}))$.[2] We denote $\mathbf{s}_{\leq t} = (\mathbf{s}_1, ..., \mathbf{s}_t)$, $\mathbf{x}_{\leq t} = (\mathbf{x}_1, ..., \mathbf{x}_t)$ and $\mathbf{x}(\mathbf{s}_{\leq t}) = (\mathbf{x}_1(\mathbf{s}_1), ..., \mathbf{x}_t(\mathbf{s}_t))$ to simplify notation.

In addition, we can extend this formulation further by allowing measurement delays to be included. Now that we have incorporated a time element, it also becomes natural that some features will take more or less time to measure than others (for example blood cultures can take up to one week to perform). To incorporate this into our formulation, we define a measurement time vector $\tau = (\tau_1, ..., \tau_d) \in \mathcal{T}^d$ which indicates the length of time it takes to measure each feature. Then in this setting, our "current" feature vector, $\mathbf{x}_t(\mathbf{s}_{\leq t})$, now

[2]This is actually no different to the static setting where $\mathbf{x}(\mathbf{s}_j)$ contains all information found in $\mathbf{x}(\mathbf{s}_1), ..., \mathbf{x}(\mathbf{s}_{j-1})$.

depends on all[3] selections made in the past (i.e. on $\mathbf{s}_{\leq t}$ rather than just $\mathbf{s}_t$) and is defined by

$$x_t(\mathbf{s}_{\leq t})^i = \begin{cases} x^i_{t-\tau_i} & \text{if } s^i_{t-\tau_i} = 1 \\ * & \text{if } s^i_{t-\tau_i} = 0 \end{cases} \tag{5.2}$$

so that if feature i was selected τ_i steps ago, then its value appears now in the current set of measured values. In this setting, we also write $\mathbf{x}(\mathbf{s}_{\leq t}) = (\mathbf{x}_1(\mathbf{s}_{\leq 1}), ..., \mathbf{x}_t(\mathbf{s}_{\leq t}))$.

The goal here is as in the static setting, where the total cost is now $\sum_{t=1}^{T} \mathbf{c}^T \mathbf{s}_t = \sum_{t=1}^{T} \sum_{i=1}^{d} c^i \times s^i_t$ and the conditional distribution constraint requires that Y_t given $\mathbf{X}_{\leq t}$ has the same distribution as Y_t given $\mathbf{X}(\mathbf{s}_{\leq t})$ for all $t \in \{1, ..., T\}$.

A time-series dataset, which we denote by $\mathcal{D}$, consists of N patient observations, assumed i.i.d. according to p so that $\mathcal{D} = \{(\mathbf{x}_{t,i}, y_{t,i})_{t=1}^{T_i}\}_{i=1}^{N}$ where $(\mathbf{x}_{t,i}, y_{t,i})_{t=1}^{T_i}$ is the stream corresponding to patient i of (random) length T_i.

In the remainder of the chapter, the more general time-series setting will be used by default. When reading the rest of the chapter, keep in mind that the discussion also applies to the static setting.

5.2.3 Optimization problem

Based on the above problem formulations, the optimization problem can be determined as follows.

$$\begin{aligned} \min_{\mathbf{s}_1, ..., \mathbf{s}_T} \quad & \sum_{t=1}^{T} \mathbb{E}_{\mathbf{x} \sim p_X} \left[\mathbf{c}^T \mathbf{s}_t \right] \\ \text{s.t.} \quad & (Y_t | \mathbf{X}_{\leq t} = \mathbf{x}_{\leq t}) \stackrel{d}{=} (Y_t | \mathbf{X}(\mathbf{s}_{\leq t}) = \mathbf{x}(\mathbf{s}_{\leq t})) \text{ for all } t \in \{1, 2, ..., T\} \end{aligned} \tag{5.3}$$

In order to find a suitable (tractable) sensing decision sequence, we transform the distributional constraint into a soft constraint using the Kullback-Leibler (KL) divergence. To do this, we consider the problem of minimizing the KL divergence between the two conditional

[3] In fact, it depends only on $\mathbf{s}_u$ for $u \in \{t - \tau_i : i = 1, ..., d\}$, i.e. the times in the past in which measurements were "started" and whose results would be reported now.

distributions with an added cost penalty term. The objective function we aim to minimize (with respect to the sensing decision sequence) is then

$$\sum_{t=1}^{T} \mathbb{E}_{\mathbf{x}\sim p_X}\Big[\big[KL((Y_t|\mathbf{X}_{\leq t}=\mathbf{x}_{\leq t})||(Y_t|\mathbf{X}(\mathbf{s}_{\leq t})=\mathbf{x}(\mathbf{s}_{\leq t}))\big)\big] + \lambda \mathbf{c}^T\mathbf{s}_t\Big] \tag{5.4}$$

where $\lambda \geq 0$ is a hyper-parameter that trades-off between the constraint (KL term) and the objective (cost term).

We can rewrite the KL divergence term as

$$\begin{aligned} &KL((Y_t|\mathbf{X}_{\leq t}=\mathbf{x}_{\leq t})||(Y_t|\mathbf{X}(\mathbf{s}_{\leq t})=\mathbf{x}(\mathbf{s}_{\leq t}))) \\ &= \int_{\mathcal{Y}} p_Y(y|\mathbf{x}_{\leq t})\Big[\log(p_Y(y|\mathbf{x}_{\leq t})) - \log(p_Y(y|\mathbf{x}(\mathbf{s}_{\leq t})))\Big]dy \end{aligned}$$

and we note that $\log(p_Y(y|\mathbf{x}_{\leq t}))$ is independent of the sensing decision sequence $\mathbf{s}_{\leq t}$. We can therefore define an equivalent loss, $l(\mathbf{x}_{\leq t}, \mathbf{s}_{\leq t})$, as follows

$$l(\mathbf{x}_{\leq t}, \mathbf{s}_{\leq t}) = \int_{\mathcal{Y}} p_Y(y|\mathbf{x}_{\leq t})\Big[-\log(p_Y(y|\mathbf{x}(\mathbf{s}_{\leq t})))\Big]dy. \tag{5.5}$$

Then, the new optimization problem is defined as

$$\min_{\mathbf{s}_1,\ldots,\mathbf{s}_T} \quad \sum_{t=1}^{T} \mathbb{E}_{\mathbf{x}\sim p_X}\Big[l(\mathbf{x}_{\leq t}, \mathbf{s}_{\leq t}) + \lambda \mathbf{c}^T\mathbf{s}_t\Big]. \tag{5.6}$$

5.3 Proposed model

In order to solve the optimization problem given in Equation (5.6), we first need to estimate the unknown density function: $p_Y(\cdot|\mathbf{x}(\mathbf{s}_{\leq t}))$. To do this, we introduce a predictor function $f_\phi : \prod_{i=1}^{t}(\mathcal{X} \times \{0,1\}^d) \to \mathcal{Y}$ parameterized by ϕ which will be trained to predict y given all (selected) observations up until time t (i.e. $\mathbf{x}(\mathbf{s}_{\leq t})$ and $\mathbf{s}_{\leq t}$).

In order to perform sensing decisions (which are binary), we introduce a selector function $f_\theta : \prod_{i=1}^{t}(\mathcal{X} \times \{0,1\}^d) \to [0,1]^d$ parameterized by θ that will output continuous values in

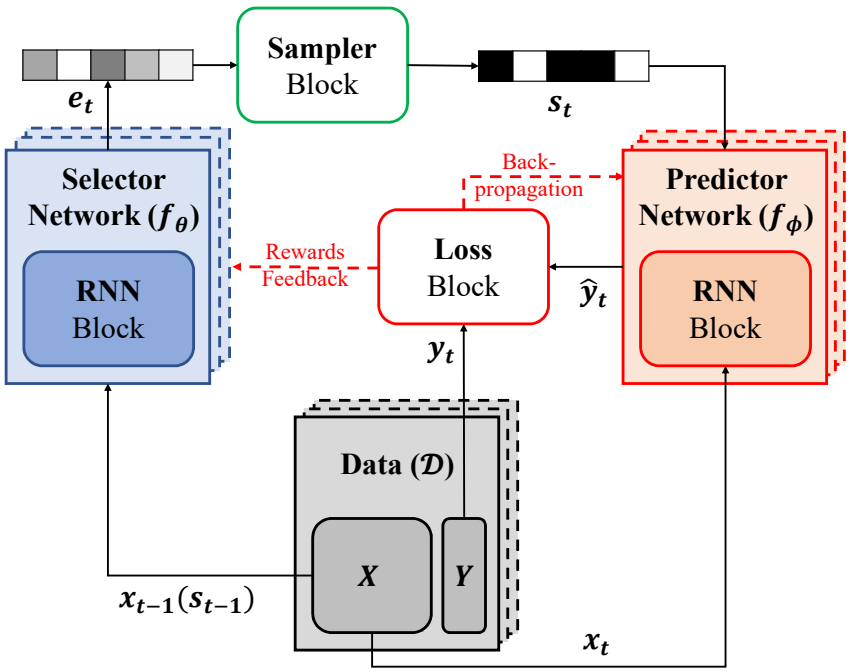

Figure 5.3: Block diagram of ASAC.

$[0,1]^d$ which will be treated as probabilities to then be sampled from to create an output in $\{0,1\}^d$. The selection mechanism is therefore *probabilistic* in nature, and as such our optimization problem in Equation (5.6) now needs to include an expectation over the sensing decision sequence $\mathbf{s}_{\leq T}$. This selector function f_θ will take measurements up until time t as input and then output probabilities from which the decision sequence for time $t+1$ will be sampled. In order to "back-propagate" through the sampling process, we draw on actor-critic models [KT00] to derive the gradient of our selector function loss in Section 5.3.2.

These two networks will be trained iteratively. This is important because both functions influence each other. The predictor function directly determines the loss of the selector function and thus has a direct impact on the training of the selector function. The selector function, on the other hand, has the more subtle effect of changing the distribution over which the predictor function needs to perform well. As the selector function is updated, the input distribution for the predictor network changes, and it is important that the predictor function performs well on the new distribution. As such, the predictor network needs to be updated after each selector function update (and vice-versa).

5.3.1 Predictor function

The predictor function is trained to minimize a prediction loss

$$\mathcal{L}(\phi) = \sum_{t=1}^{T} \mathbb{E}_{\mathbf{x}\sim p_X}[l_t(\phi)] \tag{5.7}$$

where for C-class classification we have the standard cross-entropy loss given by

$$l_t(\phi) = -\sum_{i=1}^{C} y_t^i \log(f_\phi^i(\mathbf{x}(\mathbf{s}_{\leq t}), \mathbf{s}_{\leq t})) \tag{5.8}$$

and for regression we have the standard mean-squared error loss given by

$$l_t(\phi) = (y_t - f_\phi(\mathbf{x}(\mathbf{s}_{\leq t}), \mathbf{s}_{\leq t}))^2. \tag{5.9}$$

We then use $l_t(\phi)$ as our estimate for $l(\mathbf{x}_{\leq t}, \mathbf{s}_{\leq t})$.

f_ϕ can be implemented using any function approximator capable of dealing with time-series inputs (though in the static setting it needs only to be able to deal with static inputs). In this chapter, we model f_ϕ as a Recurrent Neural Network (RNN) (in particular as an LSTM [HS97]).

We explicitly model the predictor function f_ϕ using the RNN structure as follows. At time stamp t, we first define the hidden state H_t by

$$H_t = f_1(H_{t-1}, \mathbf{s}_t, \mathbf{x}(\mathbf{s}_t))$$

where f_1 is some function parameterized as a fully connected network (the same network is used for each time point). The output of the predictor network is then given by

$$f_\phi(\mathbf{x}(\mathbf{s}_{\leq t}), \mathbf{s}_{\leq t}) = f_2(H_t) = f_2(f_1(H_{t-1}, \mathbf{s}_t, \mathbf{x}(\mathbf{s}_t)))$$

for another function f_2 parameterized as a (different) fully connected network.

Note that H_t depends on $H_{t-1}, \mathbf{s}_t$ and $\mathbf{x}(\mathbf{s}_t)$. Iterating this dependency we get that H_t depends on $\mathbf{s}_{\leq t}$ and $\mathbf{x}(\mathbf{s}_{\leq t})$.

5.3.2 Selector function

The selector function, $f_\theta : \prod_{i=1}^{t}(\mathcal{X} \times \{0,1\}^d) \to [0,1]^d$, outputs probabilities from which we sample independently to obtain a sensing decision. The probability of a given sensing decision, $\mathbf{s} = (s^1, ..., s^d)$ given the observations and selections made until time t is given by

$$\pi_\theta(\mathbf{s}|\mathbf{x}(\mathbf{s}_{\leq t}), \mathbf{s}_{\leq t}) = \prod_{i=1}^{d} f_\theta(\mathbf{x}(\mathbf{s}_{\leq t}), \mathbf{s}_{\leq t})^{s^i}(1 - f_\theta(\mathbf{x}(\mathbf{s}_{\leq t}), \mathbf{s}_{\leq t}))^{1-s^i}$$

Using a slight abuse of notation, we will write $\mathbf{s} \sim \theta$ and $\mathbf{s}_t \sim \theta|\mathbf{s}_{\leq t-1}$ to denote the marginal and conditional distribution of the sensing decision induced by the selector network (note that both of these are conditional on $\mathbf{x}_{\leq t-1}$). Using this, the objective function in Equation (5.6) can be rewritten as follows (we omit the outer expectation ($\mathbb{E}_{\mathbf{x}\sim p_X}$) due to space limitation and replace $l(\mathbf{x}_{\leq t}, \mathbf{s}_{\leq t})$ with $l_t(\phi)$):

$$\begin{aligned}
\mathcal{L}(\theta) &= \sum_{t=1}^{T} \mathbb{E}_{\mathbf{s}\sim\theta}\big[l_t(\phi) + \lambda \mathbf{c}^T \mathbf{s}_t\big] \qquad (5.10)\\
&= \sum_{t=1}^{T} \mathbb{E}_{\mathbf{s}_1\sim\theta}\Big[\cdots \mathbb{E}_{\mathbf{s}_t\sim\theta|\mathbf{s}_{\leq t-1}}\big[l_t(\phi) + \lambda \mathbf{c}^T \mathbf{s}_t\big]\Big]\\
&= \sum_{t=1}^{T} \sum_{\mathbf{s}_1\in\{0,1\}^d} \pi_\theta(\mathbf{s}_1)\Big[\sum_{\mathbf{s}_t\in\{0,1\}^d} \pi_\theta(\mathbf{s}_t|\mathbf{s}_{\leq t-1}) \times \big[l_t(\phi) + \lambda \mathbf{c}^T \mathbf{s}_t\big]\Big]\\
&= \sum_{t=1}^{T} \sum_{\mathbf{s}_{\leq t}\in\{0,1\}^{d\times t}} [\prod_{\tau=1}^{t} \pi_\theta(\mathbf{s}_\tau|\mathbf{s}_{\leq \tau-1})]\big[l_t(\phi) + \lambda \mathbf{c}^T \mathbf{s}_t\big]
\end{aligned}$$

Using ideas from actor-critic models [KT00], the gradient of this loss $\nabla_\theta \mathcal{L}(\theta)$ can be shown to be

$$\nabla_\theta \mathcal{L}(\theta) = \sum_{t=1}^{T}\sum_{j=1}^{t} \mathbb{E}_{\mathbf{s}\sim\theta}\big[[l_t(\phi) + \lambda \mathbf{c}^T \mathbf{s}_t]\nabla_\theta \log \pi_\theta(\mathbf{s}_j|\mathbf{s}_{\leq j-1})\big] \qquad (5.11)$$

where $\nabla_\theta \log \pi_\theta(\mathbf{s}_j|\mathbf{s}_{\leq j-1})$ is

$$\sum_{i=1}^{d} \Big[s_j^i \nabla_\theta \log f_\theta^i(\mathbf{x}(\mathbf{s}_{\leq j-1}), \mathbf{s}_{\leq j-1}) - (1 - s_j^i) \nabla_\theta \log f_\theta^i(\mathbf{x}(\mathbf{s}_{\leq j-1}), \mathbf{s}_{\leq j-1}) \Big]. \tag{5.12}$$

which can be directly deduced from Equation (5.10).

We explicitly model the selector function f_θ using the RNN structure as follows. At time stamp t, we first define the hidden state h_t by

$$h_t = f_3(h_{t-1}, \mathbf{s}_t, \mathbf{x}(\mathbf{s}_t))$$

where f_3 is some function parameterized as a fully connected network (the same network is used for each time point). The output of the selector network is then given by

$$f_\theta(\mathbf{x}(\mathbf{s}_{\leq t}), \mathbf{s}_{\leq t}) = f_4(h_t) = f_4(f_3(h_{t-1}, \mathbf{s}_t, \mathbf{x}(\mathbf{s}_t)))$$

for another function f_4 parameterized as a (different) fully connected network.

Note that h_t depends on $h_{t-1}, \mathbf{s}_t$ and $\mathbf{x}(\mathbf{s}_t)$. Iterating this dependency we get that h_t depends on $\mathbf{s}_{\leq t}$ and $\mathbf{x}(\mathbf{s}_{\leq t})$.

Fig. 5.3 illustrates the entire structure of ASAC. Fig. 5.4 illustrates ASAC in the time-series setting. The lower part of Fig. 5.4 depicts the selector network (e_t represents the output of f_θ at time stamp t) and the upper part of the Fig. 5.4 depicts the predictor network.

5.3.3 Training the networks

The selector and predictor networks are jointly and iteratively trained. First, the predictor network (f_ϕ) is trained to minimize the predictor loss $\mathcal{L}(\phi)$ given the sensing decisions made by the selector network (f_θ). We investigated the effect of sampling multiple sensing decisions for the same time-step and sample but found that this had very little effect on the performance. As such, when we create a mini-batch to train the predictor network with, we sample only 1

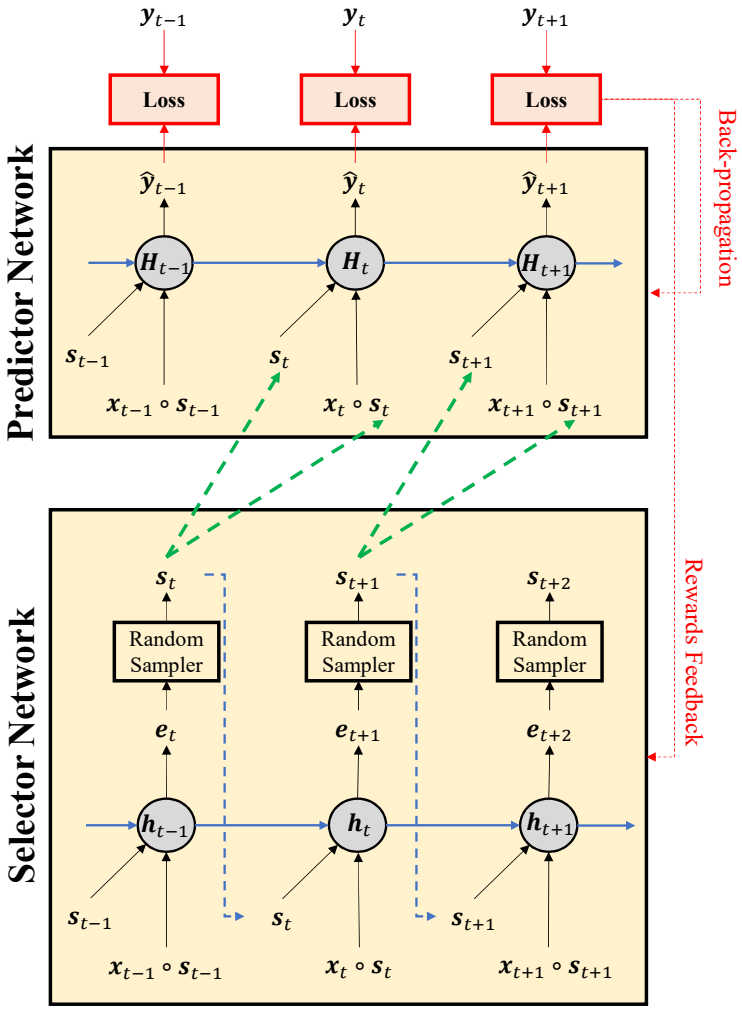

Figure 5.4: Block diagram of ASAC in a time-series setting.

sensing decision for each sample in the mini-batch.

The parameters of the predictor network are updated according to

$$\phi \leftarrow \phi - \beta \frac{1}{n_{mb}} \sum_{i=1}^{n_{mb}} \sum_{t=1}^{T_i} (y_{t,i} - f_\phi(\mathbf{x}(\mathbf{s}_{\leq t,i}), \mathbf{s}_{\leq t,i}))^2$$

where n_{mb} is the size of the mini-batch and $\beta > 0$ is the learning rate (specific to the predictor network). Then, given a fixed predictor network, the selector network parameters are updated

according to

$$\theta \leftarrow \theta - \alpha \frac{1}{n_{mb}} \sum_{i=1}^{n_{mb}} \sum_{t=1}^{T_i} \sum_{\tau=1}^{t} \Big[\big[l_{t,i}(\phi) + \lambda \mathbf{c}^T \mathbf{s}_{t,i} \big] \times \nabla_\theta \log f_\theta(\mathbf{x}(\mathbf{s}_{\leq \tau,i}), \mathbf{s}_{\leq \tau,i}) \Big]$$

where $\alpha > 0$ the learning rate (specific to the selector network). Pseudo-code of ASAC is described in Algorithm 3.

Algorithm 3 Pseudo-code of ASAC

Inputs: learning rates $\alpha, \beta > 0$, mini-batch $n_{mb} > 0$, dataset $\mathcal{D}$, hyperparameter $\lambda \geq 0$
Initialize parameters θ, ϕ
while training loss has not converged **do**
  Sample a mini-batch $(\mathbf{x}_{\leq T_i,i}, \mathbf{y}_{\leq T_i,i})_{i=1}^{n_{mb}} \sim \mathcal{D}$
  for $i = 1, ..., n_{mb}$ **do**
    for $t = 1, ..., T_i$ **do**
      Calculate selection probability vector:$\mathbf{e}_{t,i} \leftarrow f_\theta(\mathbf{x}(\mathbf{s}_{\leq t,i}), \mathbf{s}_{\leq t,i})$
      Sample selection vector from $\mathbf{e}_{t,i} : \mathbf{s}_{t,i} \sim Ber(\mathbf{e}_{t,i})$
      Calculate loss $l_{t,i}(\phi) : l_{t,i}(\phi) \leftarrow (y_{t,i} - f_\phi(\mathbf{x}(\mathbf{s}_{\leq t,i}), \mathbf{s}_{\leq t,i}))^2$
    end for
  end for
  Update the predictor network parameters ϕ

$$\phi \leftarrow \phi - \beta \frac{1}{n_{mb}} \sum_{i=1}^{n_{mb}} \sum_{t=1}^{T_i} (y_{t,i} - f_\phi(\mathbf{x}(\mathbf{s}_{\leq t,i}), \mathbf{s}_{\leq t,i}))^2$$

  Update the selector network parameters θ

$$\theta \leftarrow \theta - \alpha \frac{1}{n_{mb}} \sum_{i=1}^{n_{mb}} \sum_{t=1}^{T_i} \sum_{\tau=1}^{t} \Big[\big[l_{t,i}(\phi) + \lambda \mathbf{c}^T \mathbf{s}_{t,i} \big] \times \nabla_\theta \log f_\theta(\mathbf{x}(\mathbf{s}_{\leq \tau,i}), \mathbf{s}_{\leq \tau,i}) \Big]$$

end while=0

5.3.3.1 Missing data during training

The loss we have derived lends itself naturally to missing data in the training set. By inspecting Equations (5.11) and (5.12), we see that the gradient is made up of a sum over each feature. During training, when "back-propagating" to the selector network, for features that were selected by the network but were missing (and so their measurement can't be given), we do not back-propagate their loss. The selector network only back-propagates for

both not-selected features and selected-and-not-missing features.

5.4 Experiments

5.4.1 Data description

We use two real-world medical datasets to evaluate the performance of ASAC against Deep Sensing [YZS18a] and other baselines for various cost constraints.

ADNI dataset: The Alzheimer's Disease Neuro-imaging Initiative (ADNI) study data is a longitudinal survival dataset of per-visit measurements for 1,737 patients [PAB10]. The data tracks disease progression through clinical measurements at 1/2-year intervals, including quantitative biomarkers, cognitive tests, demographics, and risk factors. For this dataset, the adverse event we predict is unstable state occurrence.

MIMIC-III dataset: The MIMIC-III dataset [JPS16] has de-identified electronic health records (EHR) from Beth Israel Deaconess Medical Center from 2001 to 2012. It was collected from two information systems (Philips CareVue Clinical and iMDsoft MetaVision ICU) that have very different data structures. We only use data collected by MetaVision (after 2008) for consistency. We extract 40 physiological data streams from lab tests (20) and vital signs (20) that have the lowest missing rates (including heart rate, respiratory rate, blood pressures). The number of patients is 23,153 and there are 5,143 sequences of length larger than 100 time steps with the longest being 1,487 time steps. For this dataset, the adverse event we predict is death.

5.4.2 Experimental results

We evaluate the performance of ASAC against 3 benchmarks: (1) Deep Sensing [YZS18a], (2) Contextual Bandit [LCL10, AHK14], (3) Markovian Bandit [GKL11]. Furthermore, we also evaluate our model when replacing the actor-critic methodology with TD learning [Sut88] and refer to this model as ASAC with TD learning. We randomly divided the dataset into mutually exclusive training (80%) and testing (20%) sets. We conducted 10 independent

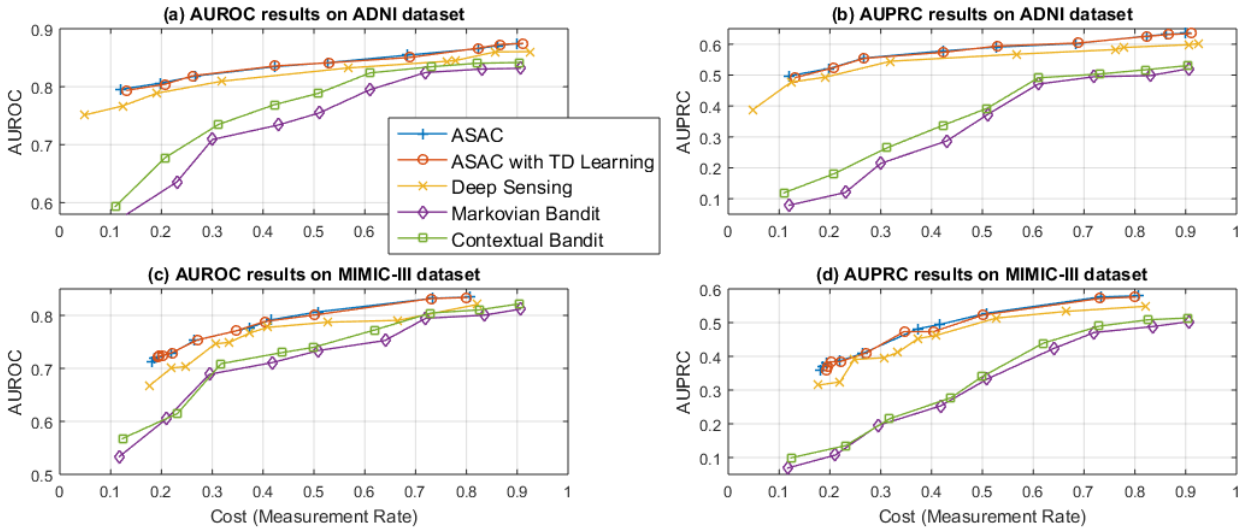

Figure 5.5: Results on risk predictions on both ADNI and MIMIC-III dataset with various cost constraints in terms of AUROC and AUPRC. X-axis is cost constraints (rate of selected measurements). Y-axis is predictive performance.

experiments with different training/testing sets in each and we report the mean and standard deviation of the performance in the 10 experiments.

In Fig. 5.5, we plot AUROC and AUPRC against the average measurement rate of all features (corresponding to all features being assigned the same cost). In MIMIC-III, we ignore the cost when a missing feature is selected.

As can be seen in Fig. 5.5, ASAC (and ASAC with TD learning) consistently outperforms all 3 benchmarks, achieving higher predictive power for the same cost across all costs. We see from Fig. 5.5(c)(d) that ASAC is robust to missing data, where we note that around 40% of the data is missing in the MIMIC-III dataset. ASAC and ASAC with TD learning achieve similar performances indicating that the ASAC framework can be robustly combined with various Reinforcement Learning frameworks to address the active sensing problem.

We can see a trade-off between accuracy and observational costs. In the ASAC framework, we can either maximize the accuracy given constraints on observational costs or minimize the cost given the desired accuracy constraint. As can be seen in the grid line in Fig. 5.5; the horizontal line represents fixing the accuracy, vertical line represents fixing the cost. We illustrate these trade-off curves for ASAC in Fig. 5.5, which shows that ASAC outperforms

state-of-the-art under both types of constraints.

5.4.3 Analysis on ASAC with synthetic datasets

We perform 3 synthetic experiments that we believe capture key attributes of an active sensing method. In each simulation, the feature distribution is a 10 dimensional auto-regressive Gaussian model over 10 time steps, i.e.

$$\mathbf{X}_t = \phi \odot \mathbf{X}_{t-1} + (1 - \phi) \odot \mathbf{Z}_t \tag{5.13}$$

where $\odot$ denotes element-wise multiplication, $\phi \in [0, 1]^{10}$ is a vector that determines the dependency of each feature on the past (a higher ϕ corresponds to a larger dependency on the past) and $\mathbf{Z}_t$ is an independent Gaussian noise vector $\mathbf{Z}_t \sim \mathcal{N}(0, \mathbf{I}^{10})$.

5.4.3.1 Time dependency vs Measurement rate

In our first experiment, we investigate the effect of time dependency on measurement rate of a variable. If we fix the cost and label-dependency of all variables to be the same, then we would expect a variable with a large ϕ to be measured less frequently by a good active sensing method (due to being more easily predicted from previous values).

To do this, we set the label, Y_t according to

$$Y_t = \exp(-0.1 \times |\sum_{i=1}^{10} X_t^i|) + \epsilon \tag{5.14}$$

where $\epsilon \sim \mathcal{N}(0, 0.1)$. We set the cost for each variable to be the same, which we vary from 1 to 5. We set $\phi = (0, 0.1, ..., 0.9)$. The measurement rate (the selection probability) of each variable is reported in Table 5.2, along with the overall RMSE for each experiment.

As can be seen in Table 5.2, ASAC meets our expectations. Features with a low ϕ, are regularly re-measured since past values are not as predictive of the present value, whereas features with a high ϕ are measured less frequently. As cost increases, we also see a monotonic

ϕ^i/Cost	1	2	3	4	5
0 ($\mathbf{X}_t^1$)	1.00	1.00	1.00	0.46	0.38
0.1 ($\mathbf{X}_t^2$)	1.00	1.00	1.00	0.44	0.36
0.2 ($\mathbf{X}_t^3$)	1.00	1.00	1.00	0.30	0.26
0.3 ($\mathbf{X}_t^4$)	1.00	1.00	1.00	0.25	0.12
0.4 ($\mathbf{X}_t^5$)	1.00	1.00	1.00	0.22	0.10
0.5 ($\mathbf{X}_t^6$)	1.00	0.98	0.23	0.21	0.07
0.6 ($\mathbf{X}_t^7$)	1.00	0.94	0.13	0.10	0.05
0.7 ($\mathbf{X}_t^8$)	1.00	0.93	0.07	0.03	0.01
0.8 ($\mathbf{X}_t^9$)	0.92	0.41	0.03	0.02	0.0
0.9 ($\mathbf{X}_t^{10}$)	0.45	0.11	0.01	0.01	0.0
RMSE	0.106	0.110	0.126	0.138	0.146

Table 5.2: Measurement rate of each feature when each feature has a different auto-regressive coefficient.

decrease in the measurement rate of all variables.

5.4.3.2 Cheaper but noisier features

In our second synthetic experiment, we investigate the effect of having cheaper, noisier versions of our original 10 features. In this experiment we are interested in understanding how well ASAC can trade-off between the cost and noise level of the noisy versions. This setting has several real-world parallels; in medicine, cheap at-home tests (such as blood pressure tests and home pregnancy tests) exist, but are less reliable (noisier) than the more expensive state-of-the-art procedures that would be used in, say, a hospital setting.

To model this, we introduce 10 new noisy features

$$\hat{\mathbf{X}}_t = \mathbf{X}_t + \delta \tag{5.15}$$

where $\delta \sim \mathcal{N}(0, \gamma)$ with $\gamma > 0$ controlling the "noisiness". In this experiment, we set the label according to

$$Y_t = \exp(-|0.1X_t^1 + 0.2X_t^2 + 0.3X_t^3 + 0.4X_t^4|) + \epsilon \tag{5.16}$$

where now we have set different magnitudes for the coefficients of the first 4 variables (and the last 6 variables are now just there as pure noise). We would expect that as we increase the cost of the true variables (or equivalently decrease the cost of the noisy variables), the variables with lower importance (X^1 and X^2) will be the first ones to be "replaced" with their noisy version, whereas it will take a higher cost for X^4 to be replaced with $\hat{X}^4$.

We fix the cost of the original features to be 1, and investigate noise levels $\gamma \in \{0.2, 0.4, 0.6\}$ and vary the cost of a noisy feature to be $\hat{c} \in \{0.1.0.2.0.5\}$. We set $\phi^i = 0.5$ for all i. In Table 5.3 we report the measurement rate of each of the first 4 variables and their noisy versions.

γ	Cost	1	0.1	1	0.2	1	0.5
	Features	$\mathbf{X}_t$	$\hat{\mathbf{X}}_t$	$\mathbf{X}_t$	$\hat{\mathbf{X}}_t$	$\mathbf{X}_t$	$\hat{\mathbf{X}}_t$
0.2	X^1	0.00	1.00	0.00	1.00	0.00	1.00
	X^2	0.00	1.00	0.00	1.00	0.29	0.70
	X^3	0.00	1.00	0.00	1.00	1.00	0.00
	X^4	0.00	1.00	0.00	1.00	1.00	0.00
0.4	X^1	0.00	1.00	0.00	0.94	1.00	0.00
	X^2	0.00	1.00	0.30	0.65	1.00	0.00
	X^3	1.00	0.00	1.00	0.00	1.00	0.00
	X^4	1.00	0.00	1.00	0.00	1.00	0.00
0.6	X^1	0.00	1.00	0.51	0.33	1.00	0.00
	X^2	0.75	0.25	1.00	0.00	1.00	0.00
	X^3	1.00	0.00	1.00	0.00	1.00	0.00
	X^4	1.00	0.00	1.00	0.00	1.00	0.00

Table 5.3: Measurement rate based on different cost and noise parameter γ for original feature ($\mathbf{X}_t$) and noisy feature ($\hat{\mathbf{X}}_t$).

As can be seen in Table 5.3, ASAC meets our expectations. As we move right and down in the table (corresponding to an increasing cost for the noisy feature and increasing noise, respectively), we see that true features are selected more frequently but that the noisy versions for the less predictive features (X^1 and X^2) are sometimes selected even at higher costs and noise levels. In particular, at $(\gamma, \hat{c}) = (0.2, 0.2)$, only the noisy features are selected. When γ is increased to 0.4, ASAC starts to select X^3 and X^4 all of the time, and X^2 some of the time, while the noisy version of X^1 is always preferred. When we increase γ to 0.6, the true version of X^2 is the only version selected by ASAC and the true version of X^1 finally becomes

desirable enough to measure (sometimes).

5.4.3.3 Y dependent cost

η	0.1		0.3		0.5	
Features	$\mathbf{X}_t$	$\hat{\mathbf{X}}_t$	$\mathbf{X}_t$	$\hat{\mathbf{X}}_t$	$\mathbf{X}_t$	$\hat{\mathbf{X}}_t$
$Y_t = 1$	0.89	0.10	0.63	0.21	0.25	0.69
$Y_t = 0$	0.13	0.81	0.14	0.80	0.12	0.78

Table 5.4: Measurement rate when the cost is different for $Y_t = 1$ and $Y_t = 0$.

In our final synthetic experiment, we allow for a cost that depends on Y. In our medical example, this could correspond to the fact that when a patient is sick, it is more important to be sure about it, than when a patient is well. In the presence of the cheaper-but-noisier features from 5.4.3.2, we expect a worsening condition to create a switch in selections. While a patient is healthy, we are happy to monitor the patient using the at-home tests, but when a patients condition appears to be deteriorating, it becomes more important that accurate measurements are made than cost being kept low.

We model this by incorporating the patients condition into the cost, setting the cost when the patient is sick ($Y_t = 1$) to be $\eta \in [0, 1]$ times the cost when the patient is healthy ($Y_t = 0$)[4]. We investigate $\eta \in \{0.1, 0.3, 0.5\}$.

We generate the true features as before, now with $\phi^i = 0.9$ for all i, and generate noisy features as in 5.4.3.2 with $\gamma = 0.4$. We set the label to be binary according to

$$Y_t = \begin{cases} 1, \text{ w.p } \exp(-0.1 \times |\sum_{i=1}^{10} X_t^i + \epsilon - 2|) \\ 0, \text{ w.p } 1 - \exp(-0.1 \times |\sum_{i=1}^{10} X_t^i + \epsilon - 2|) \end{cases}$$

where "w.p" means "with probability".

We see from Table 5.4 that ASAC is able to correctly identify that measuring the true

[4]By reducing the measurement cost, we are equivalently up-weighting the importance of accurately predicting.

features is more important when $Y_t = 1$, with measurement frequencies while $Y_t = 1$ for the true features being higher for all 3 values of η. When $\eta = 0.5$, which corresponds to the cost being half as important while the patient is sick, we see that true features are measured twice as frequently. As η decreases, and so accurate predictions become more important, we see that ASAC selects true features more frequently. When $\eta = 0.1$, ASAC selects true feature nearly 7 times more frequently while the patient is sick compared to when they are not. ASAC can therefore be used to handle settings where the trade-off between measurement cost and prediction accuracy varies according to the label (which is often the case in medicine).

5.5 Conclusion

We propose a novel active sensing framework, called Active Sensing using Actor-Critic models (ASAC), to address the important question of *what* and *when* to observe. This is critical when observations are costly. We demonstrated through real-world and synthetic experiments that the ASAC framework can significantly reduce the cost of observation with only a small loss in predictive power. Using the MIMIC-III dataset we also demonstrated that ASAC is robust to missing data.

We believe ASAC has wide-ranging applications, both in cost reduction but also for things such as planning, in which patients can be told when they might expect to need their next check-up and for what (i.e. personalized screening).

CHAPTER 6

Time-series Generative Adversarial Networks

What is a good generative model for time-series data? The temporal setting poses a unique challenge to generative modeling. A model is not only tasked with capturing the distributions of features *within* each time point, it should also capture the potentially complex dynamics of those variables *across* time. Specifically, in modeling multivariate sequential data $\mathbf{x}_{1:T} = (\mathbf{x}_1, ..., \mathbf{x}_T)$, we wish to accurately capture the conditional distribution $p(\mathbf{x}_t|\mathbf{x}_{1:t-1})$ of temporal transitions as well.

On the one hand, a great deal of work has focused on improving the temporal dynamics of autoregressive models for sequence prediction. These primarily tackle the problem of compounding errors during multi-step sampling, introducing various training-time modifications to more accurately reflect testing-time conditions [BVJ15, LGZ16, BBX16]. Autoregressive models explicitly factor the distribution of sequences into a product of conditionals $\prod_t p(\mathbf{x}_t|\mathbf{x}_{1:t-1})$. However, while useful in the context of forecasting, this approach is fundamentally deterministic, and is not truly *generative* in the sense that new sequences can be randomly sampled from them without external conditioning. On the other hand, a separate line of work has focused on directly applying the generative adversarial network (GAN) framework to sequential data, primarily by instantiating recurrent networks for the roles of generator and discriminator [Mog16, EHR17, RPB18]. While straightforward, the adversarial objective seeks to model $p(\mathbf{x}_{1:T})$ directly, without leveraging the autoregressive prior. Importantly, simply summing the standard GAN loss over sequences of vectors may not be sufficient to ensure that the dynamics of the network efficiently captures stepwise dependencies present in the training data.

In this chapter, we propose a novel mechanism to tie together both threads of research,

giving rise to a generative model explicitly trained to preserve temporal dynamics. We present Time-series Generative Adversarial Networks (TimeGAN), a natural framework for generating realistic time-series data in various domains. First, in addition to the *unsupervised* adversarial loss on both real and synthetic sequences, we introduce a stepwise *supervised* loss using the original data as supervision, thereby explicitly encouraging the model to capture the stepwise conditional distributions in the data. This takes advantage of the fact that there is more information in the training data than simply whether each datum is real or synthetic; we can expressly learn from the transition dynamics from real sequences. Second, we introduce an *embedding network* to provide a reversible mapping between features and latent representations, thereby reducing the high-dimensionality of the adversarial learning space. This capitalizes on the fact the temporal dynamics of even complex systems are often driven by fewer and lower-dimensional factors of variation. Importantly, the supervised loss is minimized by jointly training both the embedding and generator networks, such that the latent space not only serves to promote parameter efficiency—it is specifically conditioned to facilitate the generator in learning temporal relationships. Finally, we generalize our framework to handle the mixed-data setting, where both static and time-series data can be generated at the same time.

Our approach is the first to combine the flexibility of the unsupervised GAN framework with the control afforded by supervised training in autoregressive models. We demonstrate the advantages in a series of experiments on multiple real-world and synthetic datasets. Qualitatively, we conduct t-SNE [MH08] and PCA [BY95] analyses to visualize how well the generated distributions resemble the original distributions. Quantitatively, we examine how well a post-hoc classifier can distinguish between real and generated sequences. Furthermore, by applying the "train on synthetic, test on real (TSTR)" framework [EHR17, YJS19b] to the sequence prediction task, we evaluate how well the generated data preserves the predictive characteristics of the original. We find that TimeGAN achieves consistent and significant improvements over state-of-the-art benchmarks in generating realistic time-series.

6.1 Related works

TimeGAN is a generative time-series model, trained adversarially and jointly via a learned embedding space with both supervised and unsupervised losses. As such, our approach straddles the intersection of multiple strands of research, combining themes from autoregressive models for sequence prediction, GAN-based methods for sequence generation, and time-series representation learning.

Autoregressive recurrent networks trained via the maximum likelihood principle [WZ89] are prone to potentially large prediction errors when performing multi-step sampling, due to the discrepancy between *closed-loop* training (*i.e.* conditioned on ground truths) and *open-loop* inference (*i.e.* conditioned on previous guesses). Based on curriculum learning [BLC09], Scheduled Sampling was first proposed as a remedy, whereby models are trained to generate output conditioned on a mix of both previous guesses and ground-truth data [BVJ15]. Inspired by adversarial domain adaptation [GUA16], Professor Forcing involved training an auxiliary discriminator to distinguish between free-running and teacher-forced hidden states, thus encouraging the network's training and sampling dynamics to converge [LGZ16]. Actor-critic methods [KT00] have also been proposed, introducing a critic conditioned on target outputs, trained to estimate next-token value functions that guide the actor's free-running predictions [BBX16]. However, while the motivation for these methods is similar to ours in accounting for stepwise transition dynamics, they are inherently deterministic, and do not accommodate explicitly sampling from a learned distribution—central to our goal of synthetic data generation.

On the other hand, multiple studies have straightforwardly inherited the GAN framework within the temporal setting. The first (C-RNN-GAN) [Mog16] directly applied the GAN architecture to sequential data, using LSTM networks for generator and discriminator. Data is generated recurrently, taking as inputs a noise vector and the data generated from the previous time step. Recurrent Conditional GAN (RCGAN) [EHR17] took a similar approach, introducing minor architectural differences such as dropping the dependence on the previous output while conditioning on additional input [MO14]. A multitude of applied studies have

since utilized these frameworks to generate synthetic sequences in such diverse domains as text [ZGC16], finance [Sim18], biosignals [HHU18], sensor [ACS17] and smart grid data [ZKK18], as well as renewable scenarios [CWK18]. Recent work [RPB18] has proposed conditioning on time stamp information to handle irregularly sampling. However, unlike our proposed technique, these approaches rely only on the binary adversarial feedback for learning, which by itself may not be sufficient to guarantee specifically that the network efficiently captures the temporal dynamics in the training data.

Finally, representation learning in the time-series setting primarily deals with the benefits of learning compact encodings for the benefit of downstream tasks such as prediction [DL15], forecasting [LHH18], and classification [SMS15]. Other works have studied the utility of learning latent representations for purposes of pre-training [FA14], disentanglement [LM18], and interpretability [HZG17]. Meanwhile in the static setting, several works have explored the benefit of combining autoencoders with adversarial training, with objectives such as learning similarity measures [LSL16], enabling efficient inference [DBP16], as well as improving generative capability [MSJ15]—an approach that has subsequently been applied to generating discrete structures by encoding and generating entire sequences for discrimination [ZKZ17]. By contrast, our proposed method generalizes to arbitrary time-series data, incorporates stochasticity at each time step, as well as employing an embedding network to identify a lower-dimensional space for the generative model to learn the stepwise distributions and latent dynamics of the data.

Fig. 6.1(a) provides a high-level block diagram of TimeGAN, and Fig. 6.2 gives an illustrative implementation, with C-RNN-GAN and RCGAN similarly detailed. For purposes of expository and experimental comparison with existing methods, we employ a standard RNN parameterization. A table of related works is illustrated in Table 6.1.

6.2 Problem formulation

Consider the general data setting where each instance consists of two elements: static features (that do not change over time, e.g. gender), and temporal features (that occur over time,

	C-RNN-GAN [Mog16]	RCGAN [EHR17]	T-Forcing [Gra13, SMH11]	P-Forcing [LGZ16]	TimeGAN (Ours)
Stochastic	✓	✓			✓
Open-loop	✓		✓	✓	✓
Adversarial loss	✓	✓		✓	✓
Supervised loss			✓	✓	✓
Discrete features				✓	✓
Embedding space					✓
Mixed-variables					✓

Table 6.1: Summary of Related Work. (Open-loop: Previous outputs are used as conditioning information for generation at each step; Mixed-variables: Accommodates static & temporal variables).

e.g. vital signs). Let $\mathcal{S}$ be a vector space of static features, $\mathcal{X}$ of temporal features, and let $\mathbf{S} \in \mathcal{S}, \mathbf{X} \in \mathcal{X}$ be random vectors that can be instantiated with specific values denoted $\mathbf{s}$ and $\mathbf{x}$. We consider tuples of the form $(\mathbf{S}, \mathbf{X}_{1:T})$ with some joint distribution p. The length T of each sequence is also a random variable, the distribution of which—for notational convenience—we absorb into p. In the training data, let individual samples be indexed by $n \in \{1, ..., N\}$, so we can denote the training dataset $\mathcal{D} = \{(\mathbf{s}_n, \mathbf{x}_{n,1:T_n})\}_{n=1}^{N}$. Going forward, subscripts n are omitted unless explicitly required.

Our goal is to use training data $\mathcal{D}$ to learn a density $\hat{p}(\mathbf{S}, \mathbf{X}_{1:T})$ that best approximates $p(\mathbf{S}, \mathbf{X}_{1:T})$. This is a high-level objective, and—depending on the lengths, dimensionality, and distribution of the data—may be difficult to optimize in the standard GAN framework. Therefore we additionally make use of the autoregressive decomposition of the joint $p(\mathbf{S}, \mathbf{X}_{1:T}) = p(\mathbf{S}) \prod_t p(\mathbf{X}_t | \mathbf{S}, \mathbf{X}_{1:t-1})$ to focus specifically on the conditionals, yielding the complementary—and simpler—objective of learning a density $\hat{p}(\mathbf{X}_t | \mathbf{S}, \mathbf{X}_{1:t-1})$ that best approximates $p(\mathbf{X}_t | \mathbf{S}, \mathbf{X}_{1:t-1})$ at any time t.

Two Objectives. Importantly, this breaks down the sequence-level objective (matching the joint distribution) into a series of stepwise objectives (matching the conditionals). The first is global,

$$\min_{\hat{p}} D\Big(p(\mathbf{S}, \mathbf{X}_{1:T}) \Big\| \hat{p}(\mathbf{S}, \mathbf{X}_{1:T})\Big) \tag{6.1}$$

where D is some appropriate measure of distance between distributions. The second is local,

$$\min_{\hat{p}} D\Big(p(\mathbf{X}_t|\mathbf{S}, \mathbf{X}_{1:t-1}) \Big\| \hat{p}(\mathbf{X}_t|\mathbf{S}, \mathbf{X}_{1:t-1})\Big) \tag{6.2}$$

for any t. Under an ideal discriminator in the GAN framework, the former takes the form of the Jensen-Shannon divergence. Using the original data for supervision via maximum-likelihood (ML) training, the latter takes the form of the Kullback-Leibler divergence. Note that minimizing the former relies on the presence of a perfect adversary (which we may not have access to), while minimizing the latter only depends on the presence of ground-truth sequences (which we do have access to). Our target, then, will be a combination of the GAN objective (proportional to Expression 6.1) and the ML objective (proportional to Expression 6.2). As we shall see, this naturally yields a training procedure that involves the simple addition of a supervised loss to guide adversarial learning.

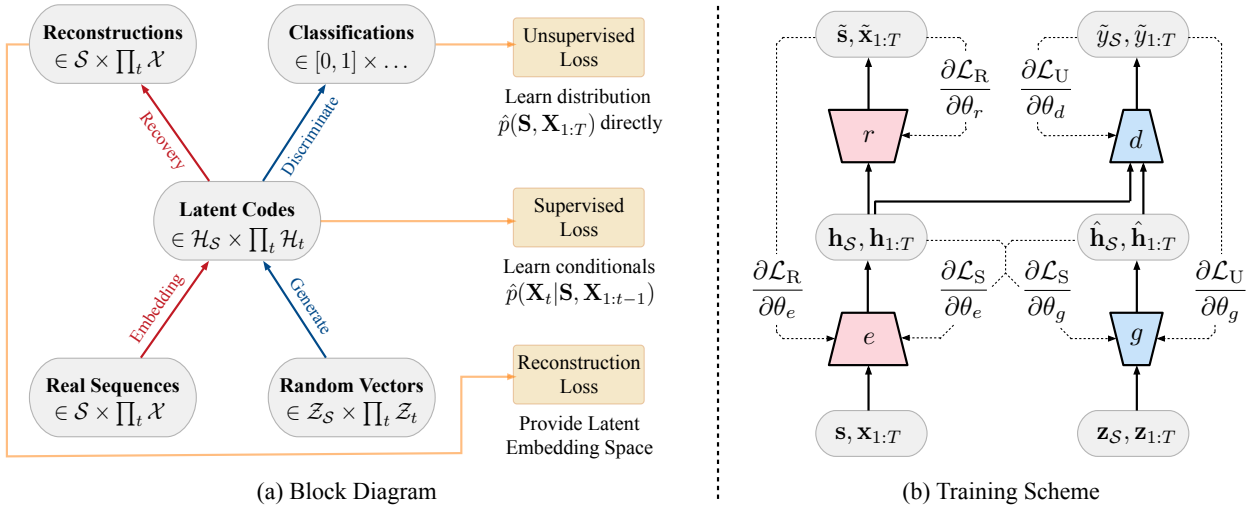

Figure 6.1: (a) Block diagram of component functions and objectives. (b) Training scheme; solid lines indicate forward propagation of data, and dashed lines indicate backpropagation of gradients.

6.3 Proposed model: Time-series GAN (TimeGAN)

TimeGAN consists of four network components: an embedding function, recovery function, sequence generator, and sequence discriminator. The key insight is that the autoencoding components (first two) are trained jointly with the adversarial components (latter two), such that TimeGAN simultaneously learns to *encode* features, *generate* representations, and *iterate* across time. The embedding network provides the latent space, the adversarial network operates within this space, and the latent dynamics of both real and synthetic data are synchronized through a supervised loss. We describe each in turn.

6.3.1 Embedding and recovery functions

The embedding and recovery functions provide mappings between feature and latent space, allowing the adversarial network to learn the underlying temporal dynamics of the data via lower-dimensional representations. Let $\mathcal{H}_{\mathcal{S}}, \mathcal{H}_{\mathcal{X}}$ denote the latent vector spaces corresponding to feature spaces $\mathcal{S}, \mathcal{X}$. Then the embedding function $e : \mathcal{S} \times \prod_t \mathcal{X} \to \mathcal{H}_{\mathcal{S}} \times \prod_t \mathcal{H}_{\mathcal{X}}$ takes static and temporal features to their latent codes $\mathbf{h}_{\mathcal{S}}, \mathbf{h}_{1:T} = e(\mathbf{s}, \mathbf{x}_{1:T})$. In this chapter, we implement e via a recurrent network,

$$\mathbf{h}_{\mathcal{S}} = e_{\mathcal{S}}(\mathbf{s}), \qquad \mathbf{h}_t = e_{\mathcal{X}}(\mathbf{h}_{\mathcal{S}}, \mathbf{h}_{t-1}, \mathbf{x}_t) \tag{6.3}$$

where $e_{\mathcal{S}} : \mathcal{S} \to \mathcal{H}_{\mathcal{S}}$ is an embedding network for static features, and $e_{\mathcal{X}} : \mathcal{H}_{\mathcal{S}} \times \mathcal{H}_{\mathcal{X}} \times \mathcal{X} \to \mathcal{H}_{\mathcal{X}}$ a recurrent embedding network for temporal features. In the opposite direction, the recovery function $r : \mathcal{H}_{\mathcal{S}} \times \prod_t \mathcal{H}_{\mathcal{X}} \to \mathcal{S} \times \prod_t \mathcal{X}$ takes static and temporal codes back to their feature representations $\tilde{\mathbf{s}}, \tilde{\mathbf{x}}_{1:T} = r(\mathbf{h}_{\mathcal{S}}, \mathbf{h}_{1:T})$. Here we implement r through a feedforward network at each step,

$$\tilde{\mathbf{s}} = r_{\mathcal{S}}(\mathbf{h}_s), \qquad \tilde{\mathbf{x}}_t = r_{\mathcal{X}}(\mathbf{h}_t) \tag{6.4}$$

where $r_{\mathcal{S}} : \mathcal{H}_{\mathcal{S}} \to \mathcal{S}$ and $r_{\mathcal{X}} : \mathcal{H}_{\mathcal{X}} \to \mathcal{X}$ are recovery networks for static and temporal embeddings. Note that the embedding and recovery functions can be parameterized by any architecture of choice, with the only stipulation being that they be autoregressive and obey

causal ordering (*i.e.* output(s) at each step can only depend on preceding information). For example, it is just as possible to implement the former with temporal convolutions [ODZ16], or the latter via an attention-based decoder [BCB14]. Here we choose implementations of Equations (6.3) and (6.4) as a minimal example to isolate the source of gains.

6.3.2 Sequence generator and discriminator

Instead of producing synthetic output directly in feature space, the generator first outputs into the embedding space. Let $\mathcal{Z}_\mathcal{S}, \mathcal{Z}_\mathcal{X}$ denote vector spaces over which known distributions are defined, and from which random vectors are drawn as input for generating into $\mathcal{H}_\mathcal{S}, \mathcal{H}_\mathcal{X}$. Then the generating function $g : \mathcal{Z}_\mathcal{S} \times \prod_t \mathcal{Z}_\mathcal{X} \to \mathcal{H}_\mathcal{S} \times \prod_t \mathcal{H}_\mathcal{X}$ takes a tuple of static and temporal random vectors to synthetic latent codes $\hat{\mathbf{h}}_\mathcal{S}, \hat{\mathbf{h}}_{1:T} = g(\mathbf{z}_\mathcal{S}, \mathbf{z}_{1:T})$. We implement g through a recurrent network,

$$\hat{\mathbf{h}}_\mathcal{S} = g_\mathcal{S}(\mathbf{z}_\mathcal{S}), \qquad \hat{\mathbf{h}}_t = g_\mathcal{X}(\hat{\mathbf{h}}_\mathcal{S}, \hat{\mathbf{h}}_{t-1}, \mathbf{z}_t) \tag{6.5}$$

where $g_\mathcal{S} : \mathcal{Z}_\mathcal{S} \to \mathcal{H}_\mathcal{S}$ is an generator network for static features, and $g_\mathcal{X} : \mathcal{H}_\mathcal{S} \times \mathcal{H}_\mathcal{X} \times \mathcal{Z}_\mathcal{X} \to \mathcal{H}_\mathcal{X}$ is a recurrent generator for temporal features. Random vector $\mathbf{z}_\mathcal{S}$ can be sampled from a distribution of choice, and $\mathbf{z}_t$ follows a stochastic process; here we use the Gaussian distribution and Wiener process respectively. Finally, the discriminator also operates from the embedding space. The discrimination function $d : \mathcal{H}_\mathcal{S} \times \prod_t \mathcal{H}_\mathcal{X} \to [0,1] \times \prod_t [0,1]$ receives the static and temporal codes, returning classifications $\tilde{y}_\mathcal{S}, \tilde{y}_{1:T} = d(\tilde{\mathbf{h}}_\mathcal{S}, \tilde{\mathbf{h}}_{1:T})$. The $\tilde{\mathbf{h}}_*$ notation denotes either real ($\mathbf{h}_*$) or synthetic ($\hat{\mathbf{h}}_*$) embeddings; similarly, the $\tilde{y}_*$ notation denotes classifications of either real (y_*) or synthetic ($\hat{y}_*$) data. Here we implement d via a bidirectional recurrent network with a feedforward output layer,

$$\tilde{y}_\mathcal{S} = d_\mathcal{S}(\tilde{\mathbf{h}}_\mathcal{S}) \qquad \tilde{y}_t = d_\mathcal{X}(\overleftarrow{\mathbf{u}}_t, \vec{\mathbf{u}}_t) \tag{6.6}$$

where $\vec{\mathbf{u}}_t = \vec{c}_\mathcal{X}(\tilde{\mathbf{h}}_\mathcal{S}, \tilde{\mathbf{h}}_t, \vec{\mathbf{u}}_{t-1})$ and $\overleftarrow{\mathbf{u}}_t = \overleftarrow{c}_\mathcal{X}(\tilde{\mathbf{h}}_\mathcal{S}, \tilde{\mathbf{h}}_t, \overleftarrow{\mathbf{u}}_{t+1})$ respectively denote the sequences of forward and backward hidden states, $\vec{c}_\mathcal{X}, \overleftarrow{c}_\mathcal{X}$ are recurrent functions, and $d_\mathcal{S}, d_\mathcal{X}$ are output

layer classification functions. Similarly, there are no restrictions on architecture beyond the generator being autoregressive; here we use a standard recurrent formulation for ease of exposition.

6.3.3 Jointly learning to encode, generate, and iterate

First, purely as a reversible mapping between feature and latent spaces, the embedding and recovery functions should enable accurate reconstructions $\tilde{\mathbf{s}}, \tilde{\mathbf{x}}_{1:T}$ of the original data $\mathbf{s}, \mathbf{x}_{1:T}$ from their latent representations $\mathbf{h}_{\mathcal{S}}, \mathbf{h}_{1:T}$. Therefore our first objective function is the *reconstruction loss*,

$$\mathcal{L}_{\mathrm{R}} = \mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}\sim p}\left[\|\mathbf{s} - \tilde{\mathbf{s}}\|_2 + \sum_t \|\mathbf{x}_t - \tilde{\mathbf{x}}_t\|_2\right] \tag{6.7}$$

In TimeGAN, the generator is exposed to two types of inputs during training. First, in pure open-loop mode, the generator—which is autoregressive—receives synthetic embeddings $\hat{\mathbf{h}}_{\mathcal{S}}, \hat{\mathbf{h}}_{1:t-1}$ (*i.e.* its own previous outputs) in order to generate the next synthetic vector $\hat{\mathbf{h}}_t$. Gradients are then computed on the *unsupervised loss*. This is as one would expect—that is,

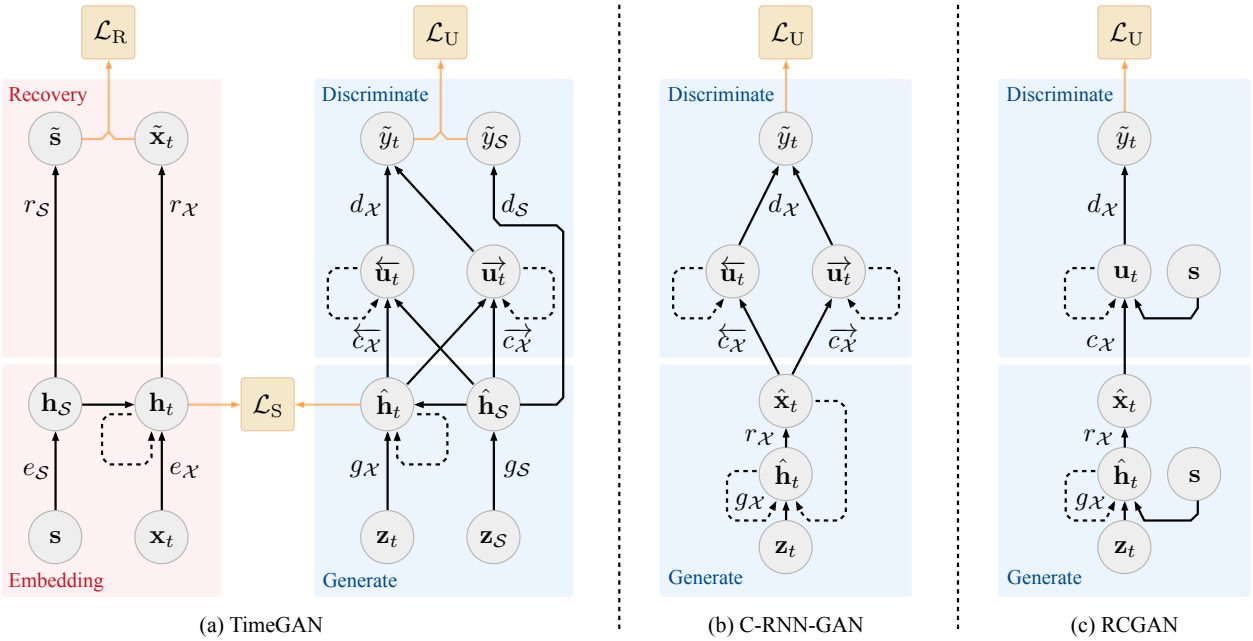

Figure 6.2: (a) TimeGAN instantiated with RNNs, (b) C-RNN-GAN, and (c) RCGAN. Solid lines denote function application, dashed lines denote recurrence, and orange lines indicate loss computation.

to allow maximizing (for the discriminator) or minimizing (for the generator) the likelihood of providing correct classifications $\hat{y}_{\mathcal{S}}, \hat{y}_{1:T}$ for both the training data $\mathbf{h}_{\mathcal{S}}, \mathbf{h}_{1:T}$ as well as for synthetic output $\hat{\mathbf{h}}_{\mathcal{S}}, \hat{\mathbf{h}}_{1:T}$ from the generator,

$$\mathcal{L}_{\mathrm{U}} = \mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}\sim p}\big[\log y_{\mathcal{S}} + \textstyle\sum_t \log y_t\big] + \mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}\sim \hat{p}}\big[\log(1-\hat{y}_{\mathcal{S}}) + \textstyle\sum_t \log(1-\hat{y}_t)\big] \tag{6.8}$$

Relying solely on the discriminator's binary adversarial feedback may not be sufficient incentive for the generator to capture the stepwise conditional distributions in the data. To achieve this more efficiently, we introduce an additional loss to further discipline learning. In an alternating fashion, we also train in closed-loop mode, where the generator receives sequences of embeddings of actual data $\mathbf{h}_{1:t-1}$ (*i.e.* computed by the embedding network) to generate the next latent vector. Gradients can now be computed on a loss that captures the discrepancy between distributions $p(\mathbf{H}_t|\mathbf{H}_{\mathcal{S}}, \mathbf{H}_{1:t-1})$ and $\hat{p}(\mathbf{H}_t|\mathbf{H}_{\mathcal{S}}, \mathbf{H}_{1:t-1})$. Applying maximum likelihood yields the familiar *supervised loss*,

$$\mathcal{L}_{\mathrm{S}} = \mathbb{E}_{\mathbf{s},\mathbf{x}_{1:T}\sim p}\big[\textstyle\sum_t \|\mathbf{h}_t - g_{\mathcal{X}}(\mathbf{h}_{\mathcal{S}}, \mathbf{h}_{t-1}, \mathbf{z}_t)\|_2\big] \tag{6.9}$$

where $g_{\mathcal{X}}(\mathbf{h}_{\mathcal{S}}, \mathbf{h}_{t-1}, \mathbf{z}_t)$ approximates $\mathbb{E}_{\mathbf{z}_t\sim\mathcal{N}}[\hat{p}(\mathbf{H}_t|\mathbf{H}_{\mathcal{S}}, \mathbf{H}_{1:t-1}, \mathbf{z}_t)]$ with one sample $\mathbf{z}_t$—as is standard in stochastic gradient descent. In sum, at any step in a training sequence, we assess the difference between the actual next-step latent vector (from the embedding function) and synthetic next-step latent vector (from the generator—conditioned on the actual historical sequence of latents). While $\mathcal{L}_{\mathrm{U}}$ pushes the generator to create realistic sequences (evaluated by an imperfect adversary), $\mathcal{L}_{\mathrm{S}}$ further ensures that it produces similar stepwise transitions (evaluated by ground-truth targets).

Optimization. Fig. 6.1(b) illustrates the mechanics of our approach at training. Let $\theta_e, \theta_r, \theta_g, \theta_d$ respectively denote the parameters of the embedding, recovery, generator, and discriminator networks. The first two components are trained on both the reconstruction and supervised losses,

$$\min_{\theta_e,\theta_r}(\lambda\mathcal{L}_{\mathrm{S}} + \mathcal{L}_{\mathrm{R}}) \tag{6.10}$$

where $\lambda \geq 0$ is a hyperparameter that balances the two losses. Importantly, $\mathcal{L}_{\mathrm{S}}$ is included such that the embedding process not only serves to reduce the dimensions of the adversarial learning space—it is actively conditioned to facilitate the generator in learning temporal relationships from the data. Next, the generator and discriminator networks are trained adversarially as follows,

$$\min_{\theta_g}(\eta\mathcal{L}_{\mathrm{S}} + \max_{\theta_d}\mathcal{L}_{\mathrm{U}}) \tag{6.11}$$

where $\eta \geq 0$ is another hyperparameter that balances the two losses. That is, in addition to the unsupervised minimax game played over classification accuracy, the generator additionally minimizes the supervised loss. By combining the objectives in this manner, TimeGAN is simultaneously trained to encode (feature vectors), generate (latent representations), and iterate (across time).

In practice, we find that TimeGAN is not sensitive to λ and η; for all experiments in Section 6.4, we set $\lambda = 1$ and $\eta = 10$. Note that while GANs in general are not known for their ease of training, we do not discover any additional complications in TimeGAN. The embedding task serves to regularize adversarial learning—which now occurs in a lower-dimensional latent space. Similarly, the supervised loss has a constraining effect on the stepwise dynamics of the generator. For both reasons, we do not expect TimeGAN to be *more* challenging to train, and standard techniques for improving GAN training are still applicable. Pseudocode of TimeGAN is described in Algorithm 4.

6.4 Experiments

Benchmarks and Evaluation. We compare TimeGAN with RCGAN [EHR17] and C-RNN-GAN [Mog16], the two most closely related methods. For purely autoregressive approaches, we compare against RNNs trained with teacher-forcing (T-Forcing) [Gra13, SMH11] as well as professor-forcing (P-Forcing) [LGZ16]. For additional comparison, we consider the performance of WaveNet [ODZ16] as well as its GAN counterpart WaveGAN [DMP18]. To assess the quality of generated data, we observe three desiderata: (1) *diversity*—samples should be distributed to cover the real data; (2) *fidelity*—samples should be indistinguishable

Algorithm 4 Pseudocode of TimeGAN

Input: $\lambda = 1$, $\eta = 10$, $\mathcal{D}$, batch size n_{mb}, learning rate γ

Initialize: $\theta_e, \theta_r, \theta_g, \theta_d$

while Not converged **do**

(1) Map between Feature Space and Latent Space

Sample $(\mathbf{s}_1, \mathbf{x}_{1,1:T_n}), ..., (\mathbf{s}_{n_{mb}}, \mathbf{x}_{n_{mb},1:T_{n_{mb}}}) \overset{\text{i.i.d.}}{\sim} \mathcal{D}$

for $n = 1, ..., n_{mb}, t = 1, ..., T_n$ **do** $(\mathbf{h}_{n,\mathcal{S}}, \mathbf{h}_{n,t}) = (e_{\mathcal{S}}(\mathbf{s}_n), e_{\mathcal{X}}(\mathbf{h}_{n,\mathcal{S}}, \mathbf{h}_{n,t-1}, \mathbf{x}_{n,t}))$

$(\tilde{\mathbf{s}}_n, \tilde{\mathbf{x}}_{n,t}) = (r_{\mathcal{S}}(\mathbf{h}_{n,\mathcal{S}}), r_{\mathcal{X}}(\mathbf{h}_{n,t}))$

(2) Generate Synthetic Latent Codes

Sample $(\mathbf{z}_{\mathcal{S},1}, \mathbf{z}_{1,1:T_n}), ..., (\mathbf{z}_{\mathcal{S},n_{mb}}, \mathbf{z}_{n_{mb},1:T_{n_{mb}}}) \overset{\text{i.i.d.}}{\sim} p_{\mathcal{Z}_{\mathcal{S}\times\mathcal{X}}}$

for $n = 1, ..., n_{mb}, t = 1, ..., T_n$ **do**

$(\hat{\mathbf{h}}_{n,\mathcal{S}}, \hat{\mathbf{h}}_{n,t}) = (g_{\mathcal{S}}(\mathbf{z}_{\mathcal{S},n}), g_{\mathcal{X}}(\hat{\mathbf{h}}_{n,\mathcal{S}}, \hat{\mathbf{h}}_{n,t-1}, \mathbf{z}_{n,t}))$

(3) Distinguish between Real and Synthetic Codes

for $n = 1, ..., n_{mb}, t = 1, ..., T_n$ **do**

$(y_{n,\mathcal{S}}, y_{n,t}) = (d_{\mathcal{S}}(\mathbf{h}_{n,\mathcal{S}}), d_{\mathcal{X}}(\vec{\mathbf{u}}_{n,t}, \vec{\mathbf{u}}_{n,t}))$

$(\hat{y}_{n,\mathcal{S}}, \hat{y}_{n,t}) = (d_{\mathcal{S}}(\hat{\mathbf{h}}_{n,\mathcal{S}}), d_{\mathcal{X}}(\hat{\vec{\mathbf{u}}}_{n,t}, \hat{\vec{\mathbf{u}}}_{n,t}))$

(4) Compute Reconstruction ($\hat{\mathcal{L}}_R$), Unsupervised ($\hat{\mathcal{L}}_U$), and Supervised ($\hat{\mathcal{L}}_S$) Losses

$\hat{\mathcal{L}}_R = \frac{1}{n_{mb}} \sum_{n=1}^{n_{mb}} \left[\|\mathbf{s}_n - \tilde{\mathbf{s}}_n\|_2 + \sum_t \|\mathbf{x}_{n,t} - \tilde{\mathbf{x}}_{n,t}\|_2 \right]$

$\hat{\mathcal{L}}_U = \frac{1}{n_{mb}} \sum_{n=1}^{n_{mb}} \left[\left[\log y_{n,\mathcal{S}} + \sum_t \log y_{n,t} \right] + \left[\log(1 - \hat{y}_{n,\mathcal{S}}) + \sum_t \log(1 - \hat{y}_{n,t}) \right] \right]$

$\hat{\mathcal{L}}_S = \frac{1}{n_{mb}} \sum_{n=1}^{n_{mb}} \left[\sum_t \|\mathbf{h}_{n,t} - g_{\mathcal{X}}(\mathbf{h}_{n,\mathcal{S}}, \mathbf{h}_{n,t-1}, \mathbf{z}_{n,t})\|_2 \right]$

(5) Update $\theta_e, \theta_r, \theta_g, \theta_d$ via Stochastic Gradient Descent (SGD)

$\theta_e = \theta_e - \gamma \nabla_{\theta_e} - \left[\lambda \hat{\mathcal{L}}_S + \hat{\mathcal{L}}_R \right]$

$\theta_r = \theta_r - \gamma \nabla_{\theta_r} - \left[\lambda \hat{\mathcal{L}}_S + \hat{\mathcal{L}}_R \right]$

$\theta_g = \theta_g - \gamma \nabla_{\theta_g} - \left[\eta \hat{\mathcal{L}}_S + \hat{\mathcal{L}}_U \right]$

$\theta_d = \theta_d + \gamma \nabla_{\theta_d} - \hat{\mathcal{L}}_U$

(6) Synthetic Data Generation

(6-1) Sample $(\mathbf{z}_{\mathcal{S},1}, \mathbf{z}_{1,1:T_n}), ..., (\mathbf{z}_{\mathcal{S},N}, \mathbf{z}_{N,1:T_N}) \overset{\text{i.i.d.}}{\sim} p_{\mathcal{Z}_{\mathcal{S}\times\mathcal{X}}}$

(6-2) Generate synthetic latent codes

for $n = 1, ..., N, t = 1, ..., T_n$ **do**

$(\hat{\mathbf{h}}_{n,\mathcal{S}}, \hat{\mathbf{h}}_{n,t}) = (g_{\mathcal{S}}(\mathbf{z}_{\mathcal{S},n}), g_{\mathcal{X}}(\hat{\mathbf{h}}_{n,\mathcal{S}}, \hat{\mathbf{h}}_{n,t-1}, \mathbf{z}_{n,t}))$

(6-3) Mapping to the feature space

for $n = 1, ..., N, t = 1, ..., T_n$ **do**

$(\hat{\mathbf{s}}_n, \hat{\mathbf{x}}_{1:T_n}) = (r_{\mathcal{S}}(\mathbf{h}_{n,\mathcal{S}}), r_{\mathcal{X}}(\mathbf{h}_{n,t}))$

Output: $\hat{\mathcal{D}} = \{\hat{\mathbf{s}}_n, \hat{\mathbf{x}}_{1:T_n}\}_{n=1}^{N}$

from the real data; and (3) *usefulness*—samples should be just as useful as the real data when used for the same predictive purposes (i.e. train-on-synthetic, test-on-real).

(1) Visualization. We apply t-SNE [MH08] and PCA [BY95] analyses on both the original and synthetic datasets (flattening the temporal dimension). This visualizes how closely the distribution of generated samples resembles that of the original in 2-dimensional space, giving a qualitative assessment of (1).

(2) Discriminative Score. For a quantitative measure of similarity, we train a post-hoc time-series classification model (by optimizing a 2-layer LSTM) to distinguish between sequences from the original and generated datasets. First, each original sequence is labeled *real*, and each generated sequence is labeled *not real*. Then, an off-the-shelf (RNN) classifier is trained to distinguish between the two classes as a standard supervised task. We then report the classification error on the held-out test set, which gives a quantitative assessment of (2).

(3) Predictive Score. In order to be useful, the sampled data should inherit the predictive characteristics of the original. In particular, we expect TimeGAN to excel in capturing conditional distributions over time. Therefore, using the synthetic dataset, we train a post-hoc sequence-prediction model (by optimizing a 2-layer LSTM) to predict next-step temporal vectors over each input sequence. Then, we evaluate the trained model on the original dataset. Performance is measured in terms of the mean absolute error (MAE); for event-based data, the MAE is computed as $|1-$ estimated probability that the event occurred$|$. This gives a quantitative assessment of (3).

All of the components (embedding network, generator, and discriminator) of TimeGAN are implemented with 3-layer GRUs with hidden dimensions 4 times the size of the input features. The dimension of the latent space is half that of the input features. We use tanh as the activation function and sigmoid as the output layer activation function such that outputs are within the $[0, 1]$ range. We also normalize the dataset to the $[0, 1]$ range using min-max scaling. We set $\lambda = 1$ and $\eta = 10$ in our experiments.

For fair comparison, we use the same underlying recurrent neural network architecture (3-layer GRUs with hidden dimensions 4 times the size of input features) for C-RNN-GAN,

RCGAN, T-Forcing, and P-Forcing as is used in TimeGAN. In the case of deterministic models (such as T-Forcing and P-Forcing), we first train an original GAN model to generate feature vectors as inputs for the initial time step, which follows the original feature distribution at the initial time step. Then, using the generated feature vector as input, we initialize the model to generate the sequence in open-loop mode. Finally, the post-hoc time-series classification and sequence-prediction models are implemented as 2-layer LSTMs with hidden dimensions 4 times the size of the input features. As before, we use tanh as the activation function and sigmoid as the output layer activation function such that outputs are within the $[0, 1]$ range. Implementation of TimeGAN can be found at `https://github.com/jsyoon0823/timegan`.

6.4.1 Illustrative example: Autoregressive Gaussian models

Our primary novelties are twofold: a supervised loss to better capture temporal dynamics, and an embedding network that provides a lower-dimensional adversarial learning space. To highlight these advantages, we experiment on sequences from autoregressive multivariate Gaussian models as follows: $\mathbf{x}_t = \phi \mathbf{x}_{t-1} + \mathbf{n}$, where $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, \sigma\mathbf{1} + (1-\sigma)\mathbf{I})$. The coefficient $\phi \in [0, 1]$ allows us to control the correlation across time steps, and $\sigma \in [-1, 1]$ controls the correlation across features.

As shown in Table 6.2, TimeGAN consistently generates higher-quality synthetic data than benchmarks, in terms of both discriminative and predictive scores. This is true across the various settings for the underlying data-generating model. Importantly, observe that the advantage of TimeGAN is greater for higher settings of temporal correlation ϕ, lending credence to the motivation and benefit of the supervised loss mechanism. Likewise, observe that the advantage of TimeGAN is also greater for higher settings of feature correlation σ, providing confirmation for the benefit of the embedding network.

6.4.2 Experiments on different types of time series data

We test the performance of TimeGAN across time-series data with a variety of different characteristics, including periodicity, discreteness, level of noise, regularity of time steps,

	Temporal Correlations (fixing $\sigma = 0.8$)			Feature Correlations (fixing $\phi = 0.8$)		
Settings	$\phi = 0.2$	$\phi = 0.5$	$\phi = 0.8$	$\sigma = 0.2$	$\sigma = 0.5$	$\sigma = 0.8$
Discriminative Score (Lower the better)						
TimeGAN	**.175±.006**	**.174±.012**	**.105±.005**	**.181±.006**	**.152±.011**	**.105±.005**
RCGAN	.177±.012	.190±.011	.133±.019	.186±.012	.190±.012	.133±.019
C-RNN-GAN	.391±.006	.227±.017	.220±.016	.198±.011	.202±.010	.220±.016
T-Forcing	.500±.000	.500±.000	.499±.001	.499±.001	.499±.001	.499±.001
P-Forcing	.498±.002	.472±.008	.396±.018	.460±.003	.408±.016	.396±.018
WaveNet	.337±.005	.235±.009	.229±.013	.217±.010	.226±.011	.229±.013
WaveGAN	.336±.011	.213±.013	.230±.023	.192±.012	.205±.015	.230±.023
Predictive Score (Lower the better)						
TimeGAN	**.640±.003**	**.412±.002**	**.251±.002**	**.282±.005**	**.261±0.002**	**.251±.002**
RCGAN	.652±.003	.435±.002	.263±.003	.292±.003	.279±.002	.263±.003
C-RNN-GAN	.696±.002	.490±.005	.299±.002	.293±.005	.280±.006	.299±.002
T-Forcing	.737±.022	.732±.012	.503±.037	.515±.034	.543±.023	.503±.037
P-Forcing	.665±.004	.571±.005	.289±.003	.406±.005	.317±.001	.289±.003
WaveNet	.718±.002	.508±.003	.321±.005	.331±.004	.297±.003	.321±.005
WaveGAN	.712±.003	.489±.001	.290±.002	.325±.003	.353±.001	.290±.002

Table 6.2: Results on autoregressive multivariate Gaussian data (Bold indicates best performance).

and correlation across time and features. The following datasets are selected on the basis of different combinations of these properties. Detailed statistics of each dataset can be found in Table 6.3.

Dataset	Sequences	Dim.	Avg. Len.	Feature Corr.	Temporal Variance	Temporal Corr.
Sines	10,000	5	24 pts	0.0117	0.3167	0.2056
Stocks	3,773	6	24 days	0.8596	0.0129	0.9902
Energy	19,711	29	24 hrs	0.2843	0.0444	0.8506
Events	149,967	54	58 events	0.0095	0.0622	0.0744

Table 6.3: Dataset statistics

(1) Sines. We simulate multivariate sinusoidal sequences of different frequencies η and phases θ, providing continuous-valued, periodic, multivariate data where each feature is independent of others. For each dimension $i \in \{1, ..., 5\}$, $x_i(t) = \sin(2\pi\eta t + \theta)$, where $\eta \sim \mathcal{U}[0, 1]$ and $\theta \sim \mathcal{U}[-\pi, \pi]$.

(2) Stocks. By contrast, sequences of stock prices are continuous-valued but aperiodic;

furthermore, features are correlated with each other. We use the daily historical Google stocks data from 2004 to 2019, including as features the volume and high, low, opening, closing, and adjusted closing prices.

(3) Energy. Next, we consider a dataset characterized by noisy periodicity, higher dimensionality, and correlated features. The UCI Appliances energy prediction dataset consists of multivariate, continuous-valued measurements including numerous temporal features measured at close intervals.

(4) Events. Finally, we consider a dataset characterized by discrete values and irregular time stamps. We use a large private lung cancer pathways dataset consisting of sequences of events and their times, and model both the one-hot encoded sequence of event types as well as the event timings.

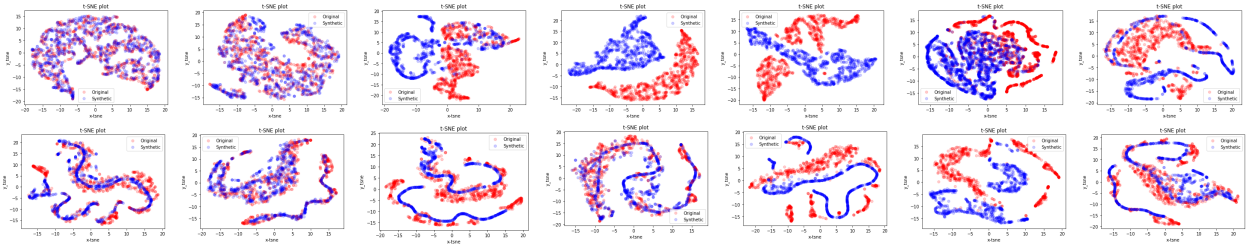

Figure 6.3: t-SNE visualization on Sines (1st row) and Stocks (2nd row). Each column provides the visualization for each of the 7 benchmarks. Red denotes original data, and blue denotes synthetic.

Visualizations with t-SNE and PCA. In Fig. 6.3, we observe that synthetic datasets generated by TimeGAN show markedly better overlap with the original data than other benchmarks using t-SNE for visualization. In fact, we (in the first column) that the blue (generated) samples and red (original) samples are almost perfectly in sync.

Discriminative and Predictive Scores. As indicated in Table 6.4, TimeGAN consistently generates higher-quality synthetic data in comparison to benchmarks on the basis of both discriminative (post-hoc classification error) and predictive (mean absolute error) scores across all datasets. For instance for Stocks, TimeGAN-generated samples achieve 0.102 which is 48% lower than the next-best benchmark (RCGAN, at 0.196)—a statistically significant improvement. Remarkably, observe that the predictive scores of TimeGAN are almost on par

with those of the original datasets themselves.

Metric	Method	Sines	Stocks	Energy	Events
Discriminative Score (Lower the Better)	TimeGAN	**.011±.008**	**.102±.021**	**.236±.012**	**.161±.018**
	RCGAN	.022±.008	.196±.027	.336±.017	.380±.021
	C-RNN-GAN	.229±.040	.399±.028	.499±.001	.462±.011
	T-Forcing	.495±.001	.226±.035	.483±.004	.387±.012
	P-Forcing	.430±.027	.257±.026	.412±.006	.489±.001
	WaveNet	.158±.011	.232±.028	.397±.010	.385±.025
	WaveGAN	.277±.013	.217±.022	.363±.012	.357±.017
Predictive Score (Lower the Better)	TimeGAN	**.093±.019**	**.038±.001**	**.273±.004**	**.303±.006**
	RCGAN	.097±.001	.040±.001	.292±.005	.345±.010
	C-RNN-GAN	.127±.004	**.038±.000**	.483±.005	.360±.010
	T-Forcing	.150±.022	**.038±.001**	.315±.005	.310±.003
	P-Forcing	.116±.004	.043±.001	.303±.006	.320±.008
	WaveNet	.117±.008	.042±.001	.311±.005	.333±.004
	WaveGAN	.134±.013	.041±.001	.307±.007	.324±.006
	Original	.094±.001	.036±.001	.250±.003	.293±.000

Table 6.4: Results on multiple time-series datasets (Bold indicates best performance).

6.4.3 Sources of gain

TimeGAN is characterized by (1) the supervised loss, (2) embedding networks, and (3) the joint training scheme. To analyze the importance of each contribution, we report the discriminative and predictive scores with the following modifications to TimeGAN: (1) without the supervised loss, (2) without the embedding networks, and (3) without jointly training the embedding and adversarial networks on the supervised loss. (The first corresponds to $\lambda = \eta = 0$, and the third to $\lambda = 0$).

We observe in Table 6.5 that all three elements make important contributions in improving the quality of the generated time-series data. The supervised loss plays a particularly important role when the data is characterized by high temporal correlations, such as in the Stocks dataset. In addition, we find that the embedding networks and joint training the with the adversarial networks (thereby aligning the targets of the two) clearly and consistently improves generative performance across the board.

Metric	Method	Sines	Stocks	Energy	Events
Discriminative Score (Lower the Better)	TimeGAN	**.011±.008**	**.102±.021**	**.236±.012**	**.161±.018**
	w/o Supervised Loss	.193±.013	.145±.023	.298±.010	.195±.013
	w/o Embedding Net.	.197±.025	.260±.021	.286±.006	.244±.011
	w/o Joint Training	.048±.011	.131±.019	.268±.012	.181±.011
Predictive Score (Lower the Better)	TimeGAN	**.093±.019**	**.038±.001**	**.273±.004**	**.303±.006**
	w/o Supervised Loss	.116±.010	.054±.001	.277±.005	.380±.023
	w/o Embedding Net.	.124±.002	.048±.001	.286±.002	.410±.013
	w/o Joint Training	.107±.008	.045±.001	.276±.004	.348±.021

Table 6.5: Source-of-gain analysis on multiple datasets (via discriminative and predictive scores).

6.5 Conclusion

In this chapter we introduce TimeGAN, a novel framework for time-series generation that combines the versatility of the unsupervised GAN approach with the control over conditional temporal dynamics afforded by supervised autoregressive models. Leveraging the contributions of the supervised loss and jointly trained embedding network, TimeGAN demonstrates consistent and significant improvements over state-of-the-art benchmarks in generating realistic time-series data.

CHAPTER 7

PATE-GAN: Generating Synthetic Data with Differential Privacy Guarantees

More and more large datasets are becoming available in a wide variety of communities. In the U.S. medical community, for example, the fraction of providers using electronic health records (EHR) increased from 9.4% in 2008 to 83.8% in 2015 [HPS17]. The availability of large datasets presents enormous opportunities for collaboration between the data-holders and the machine learning community. However, many of these large datasets, especially EHR, include sensitive information that prevents data-holders from sharing the data.

The most common way to mitigate the privacy risk of sharing sensitive records is to de-identify the records - but it is by now well-known that records that have been de-identified can be easily re-identified by linking them to other identifiable datasets [Swe97, EBT11, NS08, MS04, EN14]. (This is especially true for medical records of patients who have rare diseases.) However, if the purpose of sharing the data is to develop and validate machine learning methods for a particular task (e.g. prognostic risk scoring), real data is not necessary; it would suffice to have synthetic data that is *sufficiently like* the real data.

Precisely what this means depends on how the synthetic data will be used. For example, the synthetic data may be used to train models that will be deployed directly on real data. In this setting it is important that these methods (which we trained entirely on synthetic data) perform as well as if they had been trained on real data. Another setting to consider is one in which data-holders wish to use the synthetic data to identify the best method(s) to be used on the real data [DCJ15]. In this setting, it is not important that training on synthetic data leads to good performance on real data, but rather that comparing two methods on

the synthetic data results in conclusions similar to those that would have been drawn from comparing the two methods on the real data. We evaluate our method in both settings.

Generative Adversarial Networks (GAN) [GPM14] provide a powerful method for using real data to generate synthetic data – but it does not provide any rigorous privacy guarantees. Our method modifies the GAN machinery in a way that *does* guarantee privacy; the synthetic data is (differentially) private [DR14] with respect to the original data. To do this we modify the training procedure of the discriminator to be differentially private by using a modified version of the Private Aggregation of Teacher Ensembles (PATE) [PAE16, PSM18] framework. The Post-Processing Theorem [DR14] then guarantees that the GAN generator - which is trained only using the differentially private discriminator - will also be differentially private and thus so will the synthetic data it generates. We call our proposed framework PATE-GAN.

Using two Kaggle datasets, two different real-world medical datasets and two UCI datasets, we evaluate the utility of the samples generated by PATE-GAN in various settings with various levels of differential privacy. In line with the settings outlined above, we consider two methods for evaluating the similarity of synthetic datasets with a real dataset. The first method, first proposed in [EHR17], compares the predictive performance of models trained on the synthetic datasets and tested on the real dataset. The second method, which we propose for the first time here, compares the performance rankings of predictive models on the synthetic datasets with their performance rankings on the real dataset. We demonstrate that, for both of these methods, PATE-GAN consistently produces synthetic datasets that are "more like" the original real dataset than the synthetic datasets produced by the state-of-the-art benchmark (DPGAN [XLW18]).

The contributions of this chapter can be summarized as follows: (1) we modify the PATE framework and apply it to GANs to generate synthetic data, (2) we demonstrate in the experiments section that using PATE to enforce differential privacy results in higher quality synthetic data than DPGAN using various real-world datasets, (3) we propose a novel new metric for evaluating the generated synthetic data.

7.1 Related works

The most related previous work to this chapter is DPGAN [XLW18]. Like the proposed method, DPGAN proposes a framework for modifying the GAN framework to be differentially private, also relying on the Post-Processing Theorem to change the problem of learning a differentially private generator to learning a differentially private discriminator. Their work uses a technique introduced by [ACG16] that provides a differentially private mechanism for training deep networks. The key idea is that noise is added to the gradient of the discriminator during training to create differential privacy guarantees. These ideas are also used in [BWW17]. Our method is similar in spirit; during training of the discriminator differentially private training data is used, which results in noisy gradients, however, we use the mechanism introduced in [PAE16] which we believe gives tighter differential privacy guarantees (via tighter bounds on the effect of a single sample) than those provided in [ACG16]. This means that for the same privacy guarantees, our method is capable of producing higher quality synthetic data.

The proposed model modifies the PATE framework [PAE16, PSM18] for use in a generative model setting (specifically for use with GANs). The key to the GAN framework is that the discriminator is a *differentiable* module trained to classify samples as either real or generated. The PATE framework provides a differentially private mechanism for classification by training multiple teacher models on disjoint partitions of the data. To classify a new sample each teacher's output is evaluated on the sample and then all outputs are noisily aggregated. This noisy aggregation, though, results in a classifier that is *not* differentiable with respect to the parameters of the generator. In order to overcome this problem we follow the idea of the student model, also proposed in [PAE16], that involves taking some *public* unlabelled data, labelling it using the standard PATE mechanism and then training the student using the resulting labelled data. Because access to any public data is often an unreasonable assumption in synthetic data generation, we adapt this training paradigm in a way that does not require public data by training the student using only outputs from the (differentially private) generator.

Some previous works generate synthetic data using summary statistics of the original data [MDG16] or based on specific domain-knowledge [BBM10]; however, those methods are limited to low-dimensional feature spaces, specific fields and do not provide any differential privacy guarantees. [CBM17] generates synthetic patient records using a GAN framework. However, [CBM17] focuses only on generating discrete variables, whereas PATE-GAN is capable of generating mixed-type (continuous, discrete, and binary) variables. Furthermore, [CBM17] also does not provide any differential privacy guarantees and instead uses ad-hoc notions of privacy which are only validated empirically.

Finally, it is worth remarking that it is known to be hard in the worst-case to generate private synthetic data [UV11] and so techniques such as GANs are necessary to address this challenge.

7.2 Background

Let us denote the feature space by $\mathcal{X}$, the label space by $\mathcal{Y}$ and write $\mathcal{U} = \mathcal{X} \times \mathcal{Y}$. Let the dimension of $\mathcal{U}$ be d. Suppose that $\mathbf{X}$ and Y are random variables over $\mathcal{X}$ and $\mathcal{Y}$. We write $\mathbf{U} = (\mathbf{X}, Y)$ and $\mathbf{x}, y, \mathbf{u}$ to denote realizations of $\mathbf{X}, Y$ and $\mathbf{U}$, respectively. The dataset $\mathcal{D}$ consists of N samples of $\mathbf{u}$, assumed i.i.d. according to $\mathcal{P}_U$ denoted as $\mathcal{D} = \{\mathbf{u}_i\}_{i=1}^N = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$.

7.2.1 Differential privacy

We first provide some preliminaries on differential privacy [DR14] before describing PATE-GAN; we refer interested readers to [DR14] for a thorough exposition of differential privacy. We will denote an algorithm by $\mathcal{M}$, which takes as input a dataset $\mathcal{D}$ and outputs a value from some output space, $\mathcal{O}$.

Definition 1. *(Neighboring datasets) Two datasets $\mathcal{D}, \mathcal{D}'$ are said to be neighboring if*

$$\exists x \in \mathcal{D} \text{ s.t. } \mathcal{D} \setminus \{x\} = \mathcal{D}'. \tag{7.1}$$

Definition 2. *(Differential Privacy) A randomized algorithm, $\mathcal{M}$, is (ϵ, δ)-differentially private if for all $\mathcal{S} \subset \mathcal{O}$ and for all neighboring datasets $\mathcal{D}, \mathcal{D}'$:*

$$\mathbb{P}(\mathcal{M}(\mathcal{D}) \in \mathcal{S}) \leq e^{\epsilon}\mathbb{P}(\mathcal{M}(\mathcal{D}') \in \mathcal{S}) + \delta \tag{7.2}$$

where $\mathbb{P}$ is taken with respect to the randomness of $\mathcal{M}$.

Differential privacy provides an intuitively understandable notion of privacy - a particular sample's inclusion or exclusion in the dataset does not change the probability of a particular outcome very much: it does so by a multiplicative factor of e^{ϵ} and an additive amount, δ.

The following theorem, a proof of which can be found in [DR14], allows us to move the burden of differential privacy to the discriminator; the differential privacy of the generator will follow by the theorem.

Theorem. *(Post-processing) Let $\mathcal{M}$ be an (ϵ, δ)-differentially private algorithm and let $f : \mathcal{O} \to \mathcal{O}'$ where $\mathcal{O}'$ is any arbitrary space. Then $f \circ \mathcal{M}$ is (ϵ, δ)-differentially private.*

7.2.2 Private Aggregation of Teacher Ensembles (PATE)

In this section we describe the PATE mechanism first defined in [PAE16] and later improved upon by [PSM18]. The PATE mechanism provides a differentially private method for classification, a core component of the GAN framework; the discriminator is a classifier trained to identify whether samples are real/fake.

In order to build a differentially private classifier, the dataset is first divided into k *disjoint* subsets $\mathcal{D}_1, ..., \mathcal{D}_k$. k classifiers, $T_1, ..., T_k$ (referred to as *teachers*) are then trained *separately* on the k sub-datasets (i.e. T_i is only trained on $\mathcal{D}_i$). Given a new input feature vector $\mathbf{x}$ to classify, the differentially private output is given by passing $\mathbf{x}$ to each of the k teachers, and then performing a *noisy* aggregation of the resulting outputs.

Formally, given the k teachers, m possible classes and an input feature vector, $\mathbf{x}$, set

$$n_j(\mathbf{x}) = |\{T_i : T_i(\mathbf{x}) = j\}| \text{ for } j = 1, ..., m \tag{7.3}$$

so that $n_j(\mathbf{x})$ is the number of teachers that output class j for $\mathbf{x}$. The output of the PATE_λ mechanism for input $\mathbf{x}$ is then defined as

$$\text{PATE}_\lambda(\mathbf{x}) = \arg\max_{j \in [m]}(n_j(\mathbf{x}) + Y_j) \tag{7.4}$$

where $Y_1, ..., Y_m$ are i.i.d. $Lap(\lambda)$ random variables. The following result, found in [PAE16], follows from [DR14].

Theorem. *The output of a single query to the $PATE_\lambda$ mechanism is $(\frac{1}{\lambda}, 0)$-differentially private.*

In order to apply this framework in the GAN framework, however, we require that the discriminator be differentiable, which the *output* of this classification mechanism is not (note that accessing the internal parameters of the teachers would violate differential privacy, the only thing we have access to in this case is the output). Instead, we draw on the PATE extension (also introduced in [PAE16]) in which a *student* model is trained. This student model (after being trained) is free to access, not only its outputs given inputs but also its internal parameters. The model itself is differentially private.

Formally, the student, S, is a classifier that is trained by taking some *public, unlabelled* data, $\mathcal{P} = \{\mathbf{x}_i\}_{i=1}^K$, passing each sample, $\mathbf{x}_i$, through the (standard) PATE mechanism, to receive a differentially private label, $\hat{y}_i$, and forming a new (noisy-)teacher-labelled dataset $\hat{\mathcal{P}} = \{(\mathbf{x}_i, \hat{y}_i)\}_{i=1}^K$ on which the student is then trained.

Importantly, we can make the student differentiable - it can be modelled using any classifier, such as a neural net. Moreover, querying the student is "free" - there is no privacy cost associated with passing an input to the student and receiving an output, the only privacy cost is in acquiring the data on which to train the student. We state the following result which follows from the analysis in [PAE16].

Theorem. *The student, S, trained on the dataset $\hat{\mathcal{P}}$ where labels were generated according to the $PATE_\lambda$ mechanism using $\lambda = \frac{K}{2\epsilon}$, is $(\epsilon, 0)$-differentially private* with respect to the original data $\mathcal{D}$.

7.3 Proposed method: PATE-GAN

The proposed method builds on GAN and PATE frameworks. We replace the GAN discriminator with a PATE mechanism so that our discriminator is differentially private, but require the (differentiable) student version to allow back-propagation to the generator. We modify the implementation of the student, noting that the training paradigm presented in [PAE16] is not appropriate for this setting due to the lack of publicly available data. Before training, we partition the dataset into k subsets, $\mathcal{D}_1, ..., \mathcal{D}_k$, with $|\mathcal{D}_i| = \frac{|\mathcal{D}|}{k}$ for $\forall i$.

7.3.1 Generator

The generator, G, is as in the standard GAN framework. Formally it is a function $G(\cdot; \theta_G) : [0,1]^d \to \mathcal{U}$, parameterized by θ_G that takes random noise, $\mathbf{z} \sim \text{Unif}([0,1]^d)$, as input and outputs a vector in $\mathcal{U} = \mathcal{X} \times \mathcal{Y}$. The generator will be trained to minimize its loss with respect to the *student*-discriminator. Given n i.i.d. samples of $\text{Unif}([0,1]^d)$, $\mathbf{z}_1, ..., \mathbf{z}_n$, the empirical loss of G at θ for fixed S is defined by

$$\mathcal{L}_G(\theta_G; S) = \sum_{j=1}^{n} \log(1 - S(G(\mathbf{z}_j; \theta_G))). \tag{7.5}$$

We will denote by $\mathcal{P}_G$ the distribution induced by G over $\mathcal{U}$.

7.3.2 Discriminator

In the standard GAN framework, there is a single discriminator, D, that is trained in a directly adversarial fashion with G, where at each iteration either G is trying to improve its loss with respect to D or D is trying to improve its loss with respect to G. In our proposed model, however, we replace D with the PATE mechanism. This means we introduce k teacher-discriminators, $T^1, ..., T^k$, and a student discriminator, S. A noticeable difference is that the adversarial training is no longer symmetrical: the teachers are now being trained to improve their loss with respect to G but G is being trained to improve its loss with respect to the student S which in turn is being trained to improve its loss with respect to the teachers.

7.3.2.1 Teacher-discriminators

Formally, the teacher-discriminators (which we will refer to simply as teachers) are functions $T_1(\cdot;\theta_T^1), ..., T_k(\cdot;\theta_T^k) : \mathcal{U} \to [0,1]$ each parameterized by θ_T^i. The teachers are given either a real sample from their corresponding partition of the dataset (i.e. T_i may receive a sample from $\mathcal{D}_i$) as input or a sample from the generator. The teachers are then trained to classify them.

Given n i.i.d. samples of Unif$([0,1]^d)$, $\mathbf{z}_1, ..., \mathbf{z}_n$, we define the empirical loss of teacher i with weights θ_T^i for fixed G by

$$\mathcal{L}_T^i(\theta_T^i) = -\Big[\sum_{\mathbf{u}\in\mathcal{D}_i} \log T_i(\mathbf{u};\theta_T^i) + \sum_{j=1}^{n} \log(1 - T_i(G(\mathbf{z}_j);\theta_T^i))\Big]. \tag{7.6}$$

Each teacher is trained in the same way the discriminator is trained in a standard GAN framework, except that here the teacher only ever sees its partition of the real data.

7.3.2.2 Student-discriminators

The main innovation of the proposed model comes from our implementation of the student-discriminator (which we will refer to simply as the student) in this setting. The differential privacy guarantee provided by the standard student model is only with respect to the original data, $\mathcal{D}$, and not the public data, $\mathcal{P}$, used to train the student. In our setting, where the entire focus is on generating synthetic data *because no data is publicly available*, we must propose a novel methodology to train the student without public data.

We first note, that the student training paradigm described in [PAE16] would involve training the student using data similar to that used to train the generator - i.e. by taking an equal number of samples from each and then labelling those using the standard PATE$_\lambda$ mechanism (where here "labelling" refers to assigning them a real/fake label - not the label y present in the data). We consider the implications of training the student on teacher-labelled generated samples only.

We first observe that during training of the generator, the discriminator is only evaluated

on samples from the generator itself, and not the real data, so by training the student only on generated samples we are in fact training it on the distribution we need it to perform well on. However, we note that if the student only sees unrealistic samples from the generator (i.e. generated samples that most teachers label as fake), then the student will not contain any information that the generator can use to improve its generated samples. It is therefore important that some of the generated samples the student is trained on are realistic. We then note that if $\text{Supp}(\mathcal{P}_U) \subset \text{Supp}(\mathcal{P}_G)$ then some of the generated samples will be realistic.

In order to ensure $\text{Supp}(\mathcal{P}_U) \subset \text{Supp}(\mathcal{P}_G)$, we normalize the data into $[0,1]^d$ and then initialize the parameters of the generator randomly using Xavier initialization. It follows that $\text{Supp}(\mathcal{P}) \subset [0,1]^d \subset G([0,1]^d) = G(\text{Supp}(\mathbf{Z})) = \text{Supp}(G(\mathbf{Z}))$ when $\mathbf{Z} \sim \text{Unif}([0,1]^d)$.

We create our training data for the student by taking n i.i.d. samples of $\text{Unif}([0,1]^d)$, $\mathbf{z}_1, ..., \mathbf{z}_n$, generating n samples using the generator, $\hat{\mathbf{u}}_1, ..., \hat{\mathbf{u}}_n$ with $\hat{\mathbf{u}}_j = G(\mathbf{z}_j)$, and using the teachers to label these using PATE_λ, setting $r_j = \text{PATE}_\lambda(\hat{\mathbf{u}}_j)$. We train the student, $S(\cdot; \theta_S) : \mathcal{U} \to [0,1]$, to maximize the standard cross-entropy loss on this teacher-labelled data, i.e.

$$\mathcal{L}_S(\theta_S) = \sum_{j=1}^{n} r_j \log S(\hat{\mathbf{u}}_j; \theta_S) + (1 - r_j) \log(1 - S(\hat{\mathbf{u}}_j; \theta_S)). \tag{7.7}$$

Although a priori the above mechanism does not appear to depend on the number of teachers, it should be noted that for fixed λ, more teachers results in the teacher-labelled dataset being less noisy - the noise being added is smaller relative to the counts n_j. This introduces a trade-off - for a small number of teachers, the noise may be too large and thus render the output meaningless; with a larger number of teachers, less data can be used to train each teacher, which may also render the output meaningless, even though the noise has a smaller effect. Finding the right balance in this problem is key. In our experiments, we use d real and d generated samples to train each teacher where d is the dimension of the input space. Although the utility of a single teacher may be low, by aggregating (even noisily) the resulting classifier actually has high utility. Moreover, by using a minimal number of samples for each teacher, the effect of any individual sample on the output is small (because there are more teachers and each sample can effect at most 1 teacher) which means that our

differential privacy guarantees are tighter - if we used fewer teachers, the mechanism still assumes that, in the worst case, the presence (or absence) of a single sample can completely flip a teacher's vote and so we still need to add the same noise.

We train G, $T^1, ..., T^k$ and S iteratively[1], with each iteration of G consisting of first performing n_T updates on all teachers, then performing n_S updates of the student. We perform generator iterations until our privacy constraint, ϵ, has been reached.

Fig 7.1 and 7.2 indicate the iterative training procedure carried out by PATE-GAN; the figures correspond to a single generator update. Pseudocode for PATE-GAN can be found in Algorithm 5.

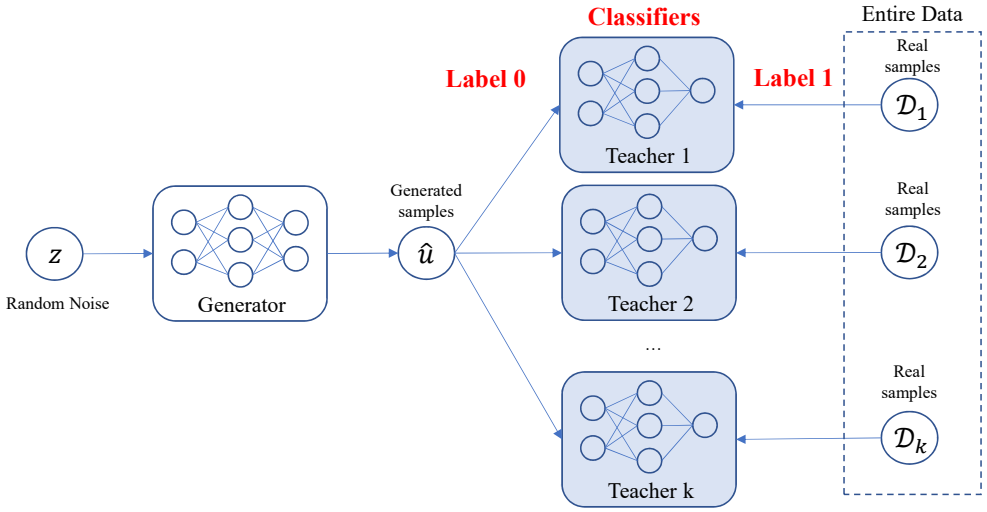

Figure 7.1: Block diagram of the training procedure for the teacher-discriminator during a single generator iteration. Teacher-discriminators are trained to minimize the classification loss when classifying samples as real samples or generated samples. During this step only the parameters of the teachers are updates (and not the generator).

To calculate the privacy of our algorithm we use the moments accountant method given in [PAE16] to derive a data-dependent privacy guarantee at run-time. We denote the moments accountant of PATE-GAN by $\alpha(l)$. The moments accountant allows us to more tightly bound the total privacy cost of our mechanism than standard composition theorems would, and moreover attributes a lower privacy cost to accessing the noisy aggregation of the teachers when the teachers have a stronger consensus with the intuition being that when the teachers

[1]The teachers can be trained in parallel.

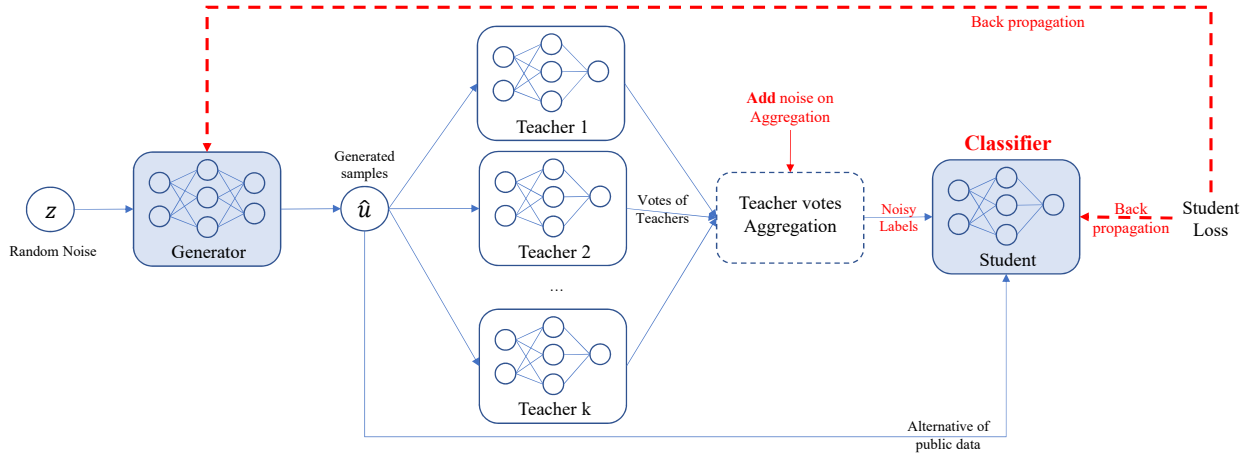

Figure 7.2: Block diagram of the training procedure for the student-discriminator and the generator. The student-discriminator is trained using noisy teacher-labelled generated samples (the noise provides the DP guarantees). The student is trained to minimize classification loss on this noisily labelled dataset, while the generator is trained to maximize the student loss. Note that the teachers are not updated during this step, only the student and the generator.

have a strong consensus, a single teacher (and therefore a single sample) has a much lower influence on the output than when the votes (n_0 and n_1) are close.

We now state the main theorem of the chapter, which follows from the theory in [PAE16].

Theorem 2. *Algorithm 5, which takes as input $\delta > 0$, a dataset, $\mathcal{D}$, and outputs G and ϵ is (ϵ, δ)-differentially private.*

The proof relies on applying the post-processing theorem where the discriminator corresponds to the mechanism $\mathcal{M}$ which takes outputs in $\mathcal{O}$ (in our case this corresponds to the weights of the discriminator), and the generator corresponds to the function f which maps from $\mathcal{O}$ to $\mathcal{O}'$ (which corresponds to the weights of the generator).

7.4 Experiments

In this section, we use a real-world Kaggle dataset (Credit card fraud detection dataset [DCJ15]) to evaluate PATE-GAN against the state-of-the-art benchmark (DPGAN [XLW18]). In addition, we provide high-level (average) results for five additional datasets (with various characteristics): MAGGIC [PAM12], UNOS-Heart wait-list [CT93], Kaggle cervical cancer

Algorithm 5 Pseudocode of PATE-GAN

Input: δ, $\mathcal{D}$, n_T, n_S, batch size n, number of teachers k, noise size λ

Initialize: θ_G, $\theta_T^1, ..., \theta_T^k$, θ_S, $\alpha(l) = 0$ for $l = 1, ..., L$

Partition dataset into k subsets $\mathcal{D}_1, ..., \mathcal{D}_k$ of size $\frac{|\mathcal{D}|}{k}$

while $\hat{\epsilon} < \epsilon$ **do**

  for $t_2 = 1, ..., n_T$ **do**

    Sample $\mathbf{z}_1, ..., \mathbf{z}_n \overset{\text{i.i.d.}}{\sim} \mathcal{P}_{\mathcal{Z}}$

    for $i = 1, ..., k$ **do**

      Sample $\mathbf{u}_1, ..., \mathbf{u}_n \overset{\text{i.i.d.}}{\sim} \mathcal{D}_i$

      Update teacher, T_i, using SGD

      $\nabla_{\theta_T^i} - \big[\sum_{j=1}^{d} \log(T_i(\mathbf{u}_j)) + \log(1 - T_i(G(\mathbf{z}_j)))\big]$

  for $t_3 = 1, ..., n_S$ **do**

    Sample $\mathbf{z}_1, ..., \mathbf{z}_n \overset{\text{i.i.d.}}{\sim} \mathcal{P}_{\mathcal{Z}}$

    for $j = 1, ..., n$ **do**

      $\hat{\mathbf{u}}_j \leftarrow G(\mathbf{z}_j)$

      $r_j \leftarrow \text{PATE}_\lambda(\hat{\mathbf{u}}_i)$ for $j = 1, ..., n$

      Update moments accountant

      $q \leftarrow \frac{2+\lambda|n_0-n_1|}{4\exp(\lambda|n_0-n_1|)}$

      for $l = 1, ..., L$ **do**

        $\alpha(l) \leftarrow \alpha(l) + \min\{2\lambda^2 l(l+1), \log((1-q)\left(\frac{1-q}{1-e^{2\lambda}q}\right)^l + qe^{2\lambda l})\}$

    Update the student, S, using SGD

    $\nabla_{\theta_S} - \sum_{j=1}^{n} r_j \log S(\hat{\mathbf{u}}_j) + (1 - r_j) \log(1 - S(\hat{\mathbf{u}}_j))$

  Sample $\mathbf{z}_1, ..., \mathbf{z}_n \overset{\text{i.i.d.}}{\sim} \mathcal{P}_{\mathcal{Z}}$

  Update the generator, G, using SGD

  $\nabla_{\theta_G}\big[\sum_{i=1}^{n} \log(1 - S(G(\mathbf{z}_i))\big]$

  $\hat{\epsilon} \leftarrow \min_l \frac{\alpha(l)+\log(\frac{1}{\delta})}{l}$

Output: G

dataset [FCF17], UCI ISOLET dataset and UCI Epileptic Seizure Recognition dataset.

7.4.1 Experimental settings

To empirically validate the quality of the generated dataset we introduce three different training-testing settings. *Setting A:* train the predictive models on the real training set, test the performance of the models on the real testing set. *Setting B:* train on the synthetic training set, test on the real testing set ([EHR17]), *Setting C:* train on the synthetic training set, test on the synthetic testing set. Note that the training set and the testing set are disjoint in both the real and synthetic datasets.

We are interested in two comparisons. If we see a high predictive performance on the real data for models that were trained on synthetic data (Setting B), we can infer that the synthetic data has captured the relationship between features and labels well. Moreover, synthetic data that does well in this setting can be used to train models without ever seeing the real data.

On the other hand, when we consider synthetic data for use in competitions such as Kaggle, we need synthetic data that allows researchers to do meaningful comparisons on the synthetic data. In this setting, the researchers will only be able to use the synthetic data as both the training and testing set, and will need to develop their algorithms using results on the synthetic data. Now it becomes important that the relative performance of two algorithms when trained and tested on the synthetic data (Setting C), is similar to their relative performance when trained and tested on the real data (Setting A). A simple requirement would be that if model 1 is better than model 2 on the real data, then model 1 is better than model 2 on the synthetic data. This allows researchers to use the synthetic data to choose the best method(s) to try on the real data (or rather to give to the data-holder to try on the real data).

For both comparisons, we use 12 different predictive models, shown in Table 7.2. We use two performance metrics to measure the capability of each model in predicting the label: (1) area under the receiver operating characteristics curve (AUROC), (2) area under the precision recall curve (AUPRC). Throughout the experiments we fix $\delta = 10^{-5}$ for use as input to PATE-GAN and DPGAN. We also report the performance of the original GAN framework

("GAN"), which serves to indicate an upper bound on performance and allows us to see how much performance is lost due to the two differential privacy mechanisms (PATE-GAN and DPGAN).

In all experiments, the depth of the generator and discriminator (student-discriminator in our case) in both PATE-GAN and the DPGAN benchmark [XLW18] is set to 3. The depth of the teacher discriminators is set to 1. The number of hidden nodes in each layer is d, $d/2$ and d (where d is the feature dimension), respectively. We use *relu* as the activation functions of each layer except for the output layer where we use the *sigmoid* activation function and the batch size is 64 for both the generator and discriminator. We set $n_T = n_S = 5$. Using cross validation, we select the number of teachers, k, among N/10 N/50 N/100 N/500 N/1000 N/5000 N/10000. The learning rate is 10^{-4} and we use Adam Optimizer to minimize the loss function.

7.4.2 Data summary and Setting A performance

Table 7.1 summarises the 6 datasets we use and provides a baseline performance for a predictive model on each dataset - Setting A refers to training and testing on the real data. The AUROC and AUPRC in this setting are upper bounds on the AUROC and AUPRC we could hope to achieve in Setting B.

Datasets	No of samples	No of features	AUROC	AUPRC
Kaggle Credit	284807	29	0.9438	0.7020
MAGGIC	30389	29	0.7069	0.3638
UNOS	23706	20	0.6416	0.6677
Kaggle Cervical cancer	858	35	0.9354	0.6314
UCI ISOLET	7797	617	0.9671	0.8758
UCI Epileptic Seizure Recognition	11500	179	0.9809	0.9511

Table 7.1: No of samples, No of features, Average AUROC and AUPRC performance across 12 different predictive models trained and tested on the real data (Setting A) for the 6 datasets: Kaggle Credit, MAGGIC, UNOS, Kaggle Cervical Cancer, UCI ISOLET, UCI Epileptic Seizure Recognition.

7.4.3 Results with Setting B

	AUROC			AUPRC		
	GAN	**PATE-GAN**	DPGAN	GAN	**PATE-GAN**	DPGAN
Logistic Regression	0.8950	0.8728	0.8720	0.4069	0.3907	0.3923
Random Forests [Bre01]	0.9075	0.8980	0.8730	0.3219	0.3157	0.2926
Gaussian Naive Bayes [Ris01]	0.8861	0.8817	0.8522	0.1963	0.1858	0.1601
Bernoulli Naive Bayes [Ris01]	0.8997	0.8968	0.8891	0.2169	0.2099	0.2069
Linear SVM [CV95]	0.7611	0.7523	0.7502	0.4473	0.4466	0.4464
Decision Tree [Qui86]	0.9102	0.9011	0.8647	0.4071	0.3978	0.3672
LDA [BNJ03]	0.8710	0.8510	0.8487	0.1956	0.1852	0.1788
AdaBoost [FS96]	0.9143	0.8952	0.8809	0.4530	0.4366	0.4234
Bagging [Bre96]	0.8951	0.8877	0.8657	0.3303	0.3221	0.3073
GBM [Fri01]	0.8848	0.8709	0.8499	0.3057	0.2974	0.2773
Multi-layer Perceptron	0.9086	0.8925	0.8787	0.4790	0.4693	0.4600
XgBoost [CG16]	0.9058	0.8904	0.8637	0.3837	0.3700	0.3440
Average	**0.8866**	**0.8737**	**0.8578**	**0.3453**	**0.3351**	**0.3219**

Table 7.2: Performance comparison of 12 different predictive models in Setting B (trained on synthetic, tested on real) in terms of AUROC and AUPRC (the generators of PATE-GAN and DPGAN are $(1, 10^{-5})$-differentially private).

In this subsection, we evaluate PATE-GAN and DPGAN in Setting B (trained on synthetic, tested on real) to understand whether or not the models are capturing the feature-label relationships well. Intuitively, if a synthetic dataset is such that a model trained on it performs well when performance is measured on real data, then the relationship between feature and label in the synthetic data is similar to that in the real data. In Table 7.2 we give the results for the Kaggle Credit dataset for all 12 predictive models. In Table 7.3, we give the performance on each dataset averaged across the 12 methods for each of the 6 datasets. Across all datasets, we see that PATE-GAN is capable of generating synthetic samples that better preserve the feature-label relationship (according to AUROC and AUPRC) than DPGAN.

We note that the performance of all models, including the original GAN model (i.e. PATE-GAN - or equivalently DPGAN - with (∞, ∞)-differential privacy) in the high dimensional UCI ISOLET and UCI Epileptic Seizure Recognition datasets is lower than in lower dimensional

Datasets	AUROC			AUPRC		
	GAN	PATE-GAN	DPGAN	GAN	PATE-GAN	DPGAN
Kaggle Credit	0.8866	0.8737	0.8578	0.3453	0.3351	0.3219
MAGGIC	0.6574	0.6446	0.6286	0.3054	0.2952	0.2820
UNOS	0.6277	0.5996	0.5552	0.6554	0.6282	0.5862
Kaggle Cervical Cancer	0.9268	0.9108	0.8699	0.5994	0.5460	0.4851
UCI ISOLET	0.8171	0.6399	0.5577	0.5561	0.2953	0.2146
UCI Epileptic Seizure Recognition	0.9173	0.7681	0.6718	0.8133	0.6512	0.5369

Table 7.3: Performance comparison of 12 different predictive models in Setting B (trained on synthetic, tested on real) in terms of AUROC and AUPRC (the generators of PATE–GAN and DPGAN are $(1, 10^{-5})$-differentially private) over 6 different datasets. GAN is (∞, ∞)-differentially private and is given to indicate an upper bound of PATE-GAN and DPGAN.

datasets. We do, however, see that both PATE-GAN and DPGAN show more significant decreases in performance than the original GAN in these high-dimensional settings. In the case of PATE-GAN, we believe this may be due to the fact that the student discriminator is trained only using data from the generator, and therefore requires that some of the generated data look somewhat realistic from the start, which is a harder requirement to satisfy as the data has more dimensions. On the other hand, in DPGAN, noise must be added to each component of the gradient (of the discriminator) and so in higher dimensions the norm of the noise added is larger. Note that in PATE-GAN, noise is added only to the teacher outputs, whose dimension (typically 1) does not depend on the dimension of the input data, and so this phenomena does not present itself in PATE-GAN. The results on both the UCI datasets would suggest that the loss from increasing noise norm (for DPGAN) is greater than from difficulty in randomly generating realistic samples (for PATE-GAN).

7.4.4 Varying the privacy constraint (ϵ)

In Fig. 7.3, we investigate the trade off between privacy constraint and utility. We report the average performance of AUROC over the 12 different predictive models for PATE-GAN and the benchmark for various ϵ (with $\delta = 10^{-5}$). As can be seen in Fig. 7.3, PATE-GAN is

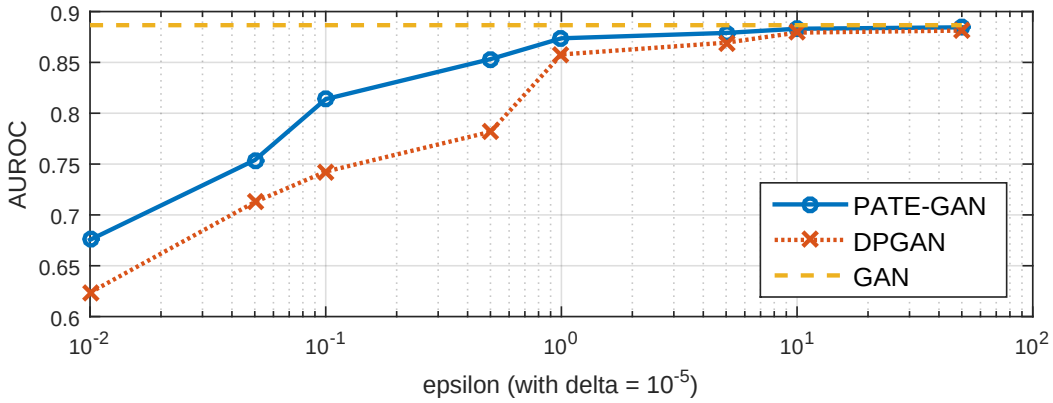

Figure 7.3: Average AUROC performance across 12 different predictive models trained on the synthetic dataset generated by PATE-GAN and DPGAN with various ϵ (with $\delta = 10^{-5}$) (Setting B).

consistently better than DPGAN over the entire range of tested ϵ. We believe this is because the PATE mechanism allows us to more tightly bound the influence of a single sample on the discriminator, and hence we can provide tighter differential privacy guarantees - when the differential privacy guarantee is fixed, this results in higher quality synthetic data. Of course, as we increase ϵ (i.e. decrease the required privacy) both methods converge to the performance of GAN and the increase in performance of PATE-GAN over DPGAN becomes smaller.

7.4.5 Setting A vs Setting C: Preserving the ranking of predictive models

As discussed at the beginning of this section, it is important that a synthetic dataset respects the ranking of models (in terms of their prediction performances) [JYS18]. To evaluate this, we now introduce a new metric, which we refer to as the Synthetic Ranking Agreement (SRA). Suppose that we have L predictive models, $f_1, f_2, ..., f_L$[2]. Furthermore, suppose that the performance of model i when trained and tested on the real data (Setting A) is $A_i \in \mathbb{R}$ and that the performance of model i when trained and tested on the synthetic data (Setting

[2]For the results in Table 7.4, we use the same 12 predictive models as used in Table 7.2

C) is $C_i \in \mathbb{R}$. Then we define the Synthetic Ranking Agreement by

$$\textbf{SRA}(\{A_i\}_{i=1}^L, \{C_i\}_{i=1}^L) = \frac{1}{L(L-1)} \sum_{j=1}^{L} \sum_{k \neq j} \mathbb{I}\Big((A_j - A_k) \times (C_j - C_k) > 0\Big) \tag{7.8}$$

where $\mathbb{I}$ is an indicator function. Note that the summand is 1 when the ordering of algorithms j and k are the same in both settings, and is 0 when the ordering in one setting differs from the ordering in the other.

	PATE-GAN	**DPGAN**		**PATE-GAN**	**DPGAN**
$\epsilon = 0.01$	0.6909	0.5273	$\epsilon = 1$	0.8364	0.8000
$\epsilon = 0.05$	0.7455	0.6909	$\epsilon = 5$	0.8909	0.8364
$\epsilon = 0.1$	0.7818	0.7455	$\epsilon = 10$	0.9091	0.8909
$\epsilon = 0.5$	0.8000	0.7818	$\epsilon = 50$	0.9091	0.9091

Table 7.4: Synthetic Ranking Probability of PATE-GAN and the benchmark when comparing Setting A and Setting C for various ϵ (with $\delta = 10^{-5}$) in terms of AUROC. The Synthetic Ranking Agreement of Original GAN is 0.9091, which is also attained by both PATE-GAN and DPGAN for $\epsilon = 50$.

We compare the SRA of PATE-GAN and the benchmark for various ϵ (with $\delta = 10^{-5}$)[3]. As can be seen in Table 7.4, PATE-GAN achieves the best SRA across all values of ϵ.

In addition, we perform a similar experiment in which we compare the ranking of features by their importance (determined by their absolute Pearson correlation coefficient with the label) on the original dataset and on the synthetic dataset (generated by PATE-GAN and the benchmark) and report the results using a metric that is identical to SRA, with the model performances ($\{A_i\}, \{C_i\}$) substituted for feature importance. As can be seen in Table 7.5, PATE-GAN achieves consistently better agreed ranking probability across all values of tested ϵ (with $\delta = 10^{-5}$).

[3]The ordering of models according to Table 7.2 is in fact quite consistent - the average agreed ranking probability (now applied to different folds of the data, rather than real vs. synthetic data) is 0.9273 (for AUROC). The rankings used are therefore sufficiently stable for this to be a meaningful metric.

	PATE-GAN	**DPGAN**		**PATE-GAN**	**DPGAN**
$\epsilon = 0.01$	0.8810	0.7963	$\epsilon = 1$	0.9048	0.8783
$\epsilon = 0.05$	0.8968	0.8148	$\epsilon = 5$	0.9127	0.8915
$\epsilon = 0.1$	0.8968	0.8333	$\epsilon = 10$	0.9153	0.8942
$\epsilon = 0.5$	0.9021	0.8545	$\epsilon = 50$	0.9153	0.9021

Table 7.5: Agreed ranking probability of PATE-GAN and the benchmark to order the features by variable importance in terms of absolute Pearson correlation coefficient

7.4.6 Quantitative analysis on the number of teachers

The number of teachers is a hyper-parameter of PATE-GAN and we choose the number of teachers among $\{N/10, N/50, N/100, N/500, N/1000, N/5000, N/10000\}$ where N is the total number of samples. As we described in the previous section, there is a trade-off between number of teachers and the corresponding quality of the synthetic data. Table 7.6 quantitatively shows the trade-off between the number of teachers and the performance (in terms of both AUROC and AUPRC).

# of teachers	$N/10$	$N/50$	$N/100$	$N/500$	**N/1000**	$N/5000$	$N/10000$
AUROC	0.5425	0.6398	0.7638	0.8343	**0.8737**	0.8655	0.8282
AUPRC	0.1273	0.2484	0.2900	0.3184	**0.3351**	0.3278	0.3092

Table 7.6: Trade-off between the number of teachers and the performances (AUROC, AUPRC)

7.5 Conclusion

In this chapter we introduced a novel methodology for generating differentially private synthetic data; we proposed modified PATE frameworks on Generative Adversarial Nets. Through several experiments we demonstrated the ability of our method to produce high quality synthetic data while being able to give strict differential privacy guarantees.

REFERENCES

[ACG16] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. "Deep learning with differential privacy." In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 308–318. ACM, 2016.

[ACS17] Moustafa Alzantot, Supriyo Chakraborty, and Mani Srivastava. "Sensegen: A deep learning architecture for synthetic sensor data generation." In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 188–193. IEEE, 2017.

[AHK14] Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert Schapire. "Taming the monster: A fast and simple algorithm for contextual bandits." In *International Conference on Machine Learning*, pp. 1638–1646, 2014.

[AHS17] Ahmed M Alaa, Scott Hu, and Mihaela van der Schaar. "Learning from Clinical Judgments: Semi-Markov-Modulated Marked Hawkes Processes for Risk Prognosis." In *Proceedings of the 34th international conference on machine learning (ICML-17)*, 2017.

[AL16] Avery Allen and Wenchen Li. *Generative Adversarial Denoising Autoencoder for Face Completion*, 2016.

[AS16] Ahmed M Alaa and Mihaela van der Schaar. "Balancing suspense and surprise: Timely decision making with endogenous information acquisition." In *Advances in Neural Information Processing Systems*, pp. 2910–2918, 2016.

[AS18a] Ahmed Alaa and Mihaela Schaar. "AutoPrognosis: Automated Clinical Prognostic Modeling via Bayesian Optimization with Structured Kernel Learning." In *International Conference on Machine Learning*, pp. 139–148, 2018.

[AS18b] Ahmed M Alaa and Mihaela van der Schaar. "A Hidden Absorbing Semi-Markov Model for Informatively Censored Temporal Data: Learning and Inference." *Journal of Machine Learning Research*, **19**(4), 2018.

[AS19] Ahmed M Alaa and Mihaela van der Schaar. "Demystifying Black-box Models with Symbolic Metamodels." In *Advances in Neural Information Processing Systems*, pp. 11301–11311, 2019.

[AYH18] Ahmed M Alaa, Jinsung Yoon, Scott Hu, and Mihaela van der Schaar. "Personalized risk scoring for critical care prognosis using mixtures of Gaussian processes." *IEEE Transactions on Biomedical Engineering*, **65**(1):207–218, 2018.

[BBM10] Anna L Buczak, Steven Babin, and Linda Moniz. "Data-driven approach for creating synthetic electronic medical records." *BMC medical informatics and decision making*, **10**(1):59, 2010.

[BBM15] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation." *PloS one*, **10**(7):e0130140, 2015.

[BBX16] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. "An actor-critic algorithm for sequence prediction." *arXiv preprint arXiv:1607.07086*, 2016.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *arXiv preprint arXiv:1409.0473*, 2014.

[BG11] Stef Buuren and Karin Groothuis-Oudshoorn. "mice: Multivariate imputation by chained equations in R." *Journal of statistical software*, **45**(3), 2011.

[BKD92] Paul R Billings, Mel A Kohn, Margaret De Cuevas, Jonathan Beckwith, Joseph S Alper, and Marvin R Natowicz. "Discrimination as a consequence of genetic testing." *American journal of human genetics*, **50**(3):476, 1992.

[BLC09] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. "Curriculum learning." In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48. ACM, 2009.

[BM99] John Barnard and Xiao-Li Meng. "Applications of multiple imputation in medical studies: from AIDS to NHANES." *Statistical methods in medical research*, **8**(1):17–36, 1999.

[BNJ03] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation." *Journal of machine Learning research*, **3**(Jan):993–1022, 2003.

[Bre96] Leo Breiman. "Bagging predictors." *Machine learning*, **24**(2):123–140, 1996.

[Bre01] Leo Breiman. "Random forests." *Machine learning*, **45**(1):5–32, 2001.

[BSH10] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert MÃžller. "How to explain individual classification decisions." *Journal of Machine Learning Research*, **11**(Jun):1803–1831, 2010.

[BVJ15] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. "Scheduled sampling for sequence prediction with recurrent neural networks." In *Advances in Neural Information Processing Systems*, pp. 1171–1179, 2015.

[BWR13] Stephen Burgess, Ian R White, Matthieu Resche-Rigon, and Angela M Wood. "Combining multiple imputation and meta-analysis with individual participant data." *Statistics in medicine*, **32**(26):4499–4514, 2013.

[BWW17] Brett K Beaulieu-Jones, Zhiwei Steven Wu, Chris Williams, and Casey S Greene. "Privacy-preserving generative deep neural networks support clinical data sharing." *BioRxiv*, p. 159756, 2017.

[BY95] Fred B Bryant and Paul R Yarnold. "Principal-components analysis and exploratory and confirmatory factor analysis." 1995.

[CBM17] Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F Stewart, and Jimeng Sun. "Generating Multi-label Discrete Patient Records using Generative Adversarial Networks." In *Machine Learning for Healthcare Conference*, pp. 286–305, 2017.

[CBS16] Edward Choi, Mohammad Taha Bahadori, Andy Schuetz, Walter F Stewart, and Jimeng Sun. "Doctor ai: Predicting clinical events via recurrent neural networks." In *Machine Learning for Healthcare Conference*, pp. 301–318, 2016.

[CFJ16] Emmanuel Jean Candès, Yingying Fan, Lucas Janson, and Jinchi Lv. *Panning for gold: Model-free knockoffs for high-dimensional controlled variable selection.* Department of Statistics, Stanford University, 2016.

[CG16] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system." In *Proceedings of the 22nd ACM international conference on knowledge discovery and data mining*, pp. 785–794. ACM, 2016.

[CPC18] 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):1–12, 2018.

[CR09] Emmanuel J Candès and Benjamin Recht. "Exact matrix completion via convex optimization." *Foundations of Computational mathematics*, **9**(6):717, 2009.

[CSW18] Jianbo Chen, Le Song, Martin Wainwright, and Michael Jordan. "Learning to Explain: An Information-Theoretic Perspective on Model Interpretation." In *International Conference on Machine Learning*, pp. 883–892, 2018.

[CT93] J Michael Cecka and Paul I Terasaki. "The UNOS scientific renal transplant registry." *Clinical transplants*, pp. 1–18, 1993.

[CV95] Corinna Cortes and Vladimir Vapnik. "Support-vector networks." *Machine learning*, **20**(3):273–297, 1995.

[CWK18] Yize Chen, Yishen Wang, Daniel Kirschen, and Baosen Zhang. "Model-free renewable scenario generation using generative adversarial networks." *IEEE Transactions on Power Systems*, **33**(3):3265–3275, 2018.

[DBP16] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. "Adversarially learned inference." *arXiv preprint arXiv:1606.00704*, 2016.

[DCI16] Yi Deng, Changgee Chang, Moges Seyoum Ido, and Qi Long. "Multiple imputation for general missing data patterns in the presence of high-dimensional data." *Scientific reports*, **6**:21689, 2016.

[DCJ15] Andrea Dal Pozzolo, Olivier Caelen, Reid A Johnson, and Gianluca Bontempi. "Calibrating probability with undersampling for unbalanced classification." In *Computational Intelligence, 2015 IEEE Symposium Series on*, pp. 159–166. IEEE, 2015.

[DHS06] A Rogier T Donders, Geert JMG van der Heijden, Theo Stijnen, and Karel GM Moons. "A gentle introduction to imputation of missing values." *Journal of clinical epidemiology*, **59**(10):1087–1091, 2006.

[DL15] Andrew M Dai and Quoc V Le. "Semi-supervised sequence learning." In *Advances in neural information processing systems*, pp. 3079–3087, 2015.

[DMP18] Chris Donahue, Julian McAuley, and Miller Puckette. "Adversarial audio synthesis." *arXiv preprint arXiv:1802.04208*, 2018.

[DR14] Cynthia Dwork, Aaron Roth, et al. "The algorithmic foundations of differential privacy." *Foundations and Trends® in Theoretical Computer Science*, **9**(3–4):211–407, 2014.

[DSZ16] Anupam Datta, Shayak Sen, and Yair Zick. "Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems." In *Security and Privacy (SP), 2016 IEEE Symposium on*, pp. 598–617. IEEE, 2016.

[EAO07] Michael E Ezzie, Scott K Aberegg, and James M O'Brien Jr. "Laboratory testing in the intensive care unit." *Critical care clinics*, **23**(3):435–465, 2007.

[EBT11] Khaled El Emam, David Buckeridge, Robyn Tamblyn, Angelica Neisa, Elizabeth Jonker, and Aman Verma. "The re-identification risk of Canadians from longitudinal demographics." *BMC medical informatics and decision making*, **11**(1):46, 2011.

[EHR17] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. "Real-valued (medical) time series generation with recurrent conditional gans." *arXiv preprint arXiv:1706.02633*, 2017.

[EN14] Yaniv Erlich and Arvind Narayanan. "Routes for breaching and protecting genetic privacy." *Nature Reviews Genetics*, **15**(6):409–421, 2014.

[FA14] Otto Fabius and Joost R van Amersfoort. "Variational recurrent auto-encoders." *arXiv preprint arXiv:1412.6581*, 2014.

[FCF17] Kelwin Fernandes, Jaime S Cardoso, and Jessica Fernandes. "Transfer learning with partial observability applied to cervical cancer screening." In *Iberian conference on pattern recognition and image analysis*, pp. 243–250. Springer, 2017.

[Fri01] Jerome H Friedman. "Greedy function approximation: a gradient boosting machine." *Annals of statistics*, pp. 1189–1232, 2001.

[FS96] Yoav Freund, Robert E Schapire, et al. "Experiments with a new boosting algorithm." In *Icml*, volume 96, pp. 148–156. Bari, Italy, 1996.

[GB96] Francois Gingras and Y Bengio. "Recurrent neural networks for missing or asynchronous data." In *NIPS*, volume 8, 1996.

[GE03] Isabelle Guyon and André Elisseeff. "An introduction to variable and feature selection." *Journal of machine learning research*, **3**(Mar):1157–1182, 2003.

[GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. "Extremely randomized trees." *Machine learning*, **63**(1):3–42, 2006.

[GG16] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning." In *international conference on machine learning*, pp. 1050–1059, 2016.

[GH16] Bobby Gheorghiu and Simon Hagens. "Measuring interoperable EHR adoption and maturity: a Canadian example." *BMC medical informatics and decision making*, **16**(1):8, 2016.

[GKL11] Yi Gai, Bhaskar Krishnamachari, and Mingyan Liu. "On the combinatorial multi-armed bandit problem with Markovian rewards." In *2011 IEEE Global Telecommunications Conference-GLOBECOM 2011*, pp. 1–6. IEEE, 2011.

[GPH00] John K Gohagan, Philip C Prorok, Richard B Hayes, and Barnett-S Kramer. "The Prostate, Lung, Colorectal and Ovarian (PLCO) cancer screening trial of the National Cancer Institute: history, organization, and status." *Controlled clinical trials*, **21**(6):251S–272S, 2000.

[GPM14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." In *Advances in Neural information processing systems*, pp. 2672–2680, 2014.

[Gra13] Alex Graves. "Generating sequences with recurrent neural networks." *arXiv preprint arXiv:1308.0850*, 2013.

[GS05] Alex Graves and Jürgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." *Neural Networks*, **18**(5):602–610, 2005.

[GSF10] Pedro J García-Laencina, José-Luis Sancho-Gómez, and Aníbal R Figueiras-Vidal. "Pattern classification with missing data: a review." *Neural Computing and Applications*, **19**(2):263–282, 2010.

[GUA16] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. "Domain-adversarial training of neural networks." *The Journal of Machine Learning Research*, **17**(1):2096–2030, 2016.

[GW18] Lovedeep Gondara and Ke Wang. "Mida: Multiple imputation using denoising autoencoders." In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 260–272. Springer, 2018.

[Hal99] Mark Andrew Hall. "Correlation-based feature selection for machine learning." 1999.

[HGD17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask r-cnn." In *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[HHU18] Shota Haradal, Hideaki Hayashi, and Seiichi Uchida. "Biosignal Data Augmentation Based on Generative Adversarial Networks." In *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 368–371. IEEE, 2018.

[HPS17] J Henry, Y Pylypchuk, T Searcy, and V Patel. "Adoption of Electronic Health Record Systems among US Non-Federal Acute Care Hospitals: 2008–2015. May 2016." *Accessed via:¡ https://dashboard. healthit. gov/evaluations/data-briefs/non-federal-acute-care-hospital-ehr-adoption-2008-2015. php¿. Accessed on June*, **21**, 2017.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." *Neural computation*, **9**(8):1735–1780, 1997.

[HZG17] Wei-Ning Hsu, Yu Zhang, and James Glass. "Unsupervised learning of disentangled and interpretable representations from sequential data." In *Advances in neural information processing systems*, pp. 1878–1889, 2017.

[HZR16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167*, 2015.

[JGP16] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical reparameterization with gumbel-softmax." *arXiv preprint arXiv:1611.01144*, 2016.

[JPS16] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. "MIMIC-III, a freely accessible critical care database." *Scientific data*, **3**:160035, 2016.

[JYS18] James Jordon, Jinsung Yoon, and Mihaela van der Schaar. "Measuring the quality of Synthetic data for use in competitions." *arXiv preprint arXiv:1806.11345*, 2018.

[KBR09] Hèlen Koch, Marloes A van Bokhoven, Gerben ter Riet, JM Tineke van Alphen-Jager, Trudy van der Weijden, Geert-Jan Dinant, and Patrick JE Bindels. "Ordering blood tests for patients with unexplained fatigue in general practice: what does it yield? Results of the VAMPIRE trial." *Br J Gen Pract*, **59**(561):e93–e100, 2009.

[KJC17] Han-Gyu Kim, Gil-Jin Jang, Ho-Jin Choi, Minho Kim, Young-Won Kim, and Jaehun Choi. "Recurrent neural networks with missing information imputation for medical examination data prediction." In *Big Data and Smart Computing (BigComp), 2017 IEEE International Conference on*, pp. 317–323. IEEE, 2017.

[KJP17] Young-Gun Kim, Kyoungwon Jung, Young-Taek Park, Dahye Shin, Soo Yeon Cho, Dukyong Yoon, and Rae Woong Park. "Rate of electronic health record adoption in South Korea: A nation-wide survey." *International journal of medical informatics*, **101**:100–107, 2017.

[KL12] David M Kreindler and Charles J Lumsden. "The effects of the irregular sample and missing data in time series analysis." *Nonlinear Dynamical Systems Analysis for the Behavioral Sciences Using Real Data*, p. 135, 2012.

[KLA15] David P Kao, James D Lewsey, Inder S Anand, Barry M Massie, Michael R Zile, Peter E Carson, Robert S McKelvie, Michel Komajda, John JV McMurray, and JoAnn Lindenfeld. "Characterization of subgroups of heart failure patients with preserved ejection fraction with possible implications for prognosis and treatment response." *European journal of heart failure*, **17**(9):925–935, 2015.

[KNS08] Kanya Kumwilaisak, Alberto Noto, Ulrich H Schmidt, Clare I Beck, Claudia Crimi, Kent Lewandrowski, and Luca M Bigatello. "Effect of laboratory testing guidelines on the utilization of tests and order entries in a surgical intensive care unit." *Critical care medicine*, **36**(11):2993–2999, 2008.

[KR92] Kenji Kira and Larry A Rendell. "A practical approach to feature selection." In *Machine Learning Proceedings 1992*, pp. 249–256. Elsevier, 1992.

[KSM16] Pieter-Jan Kindermans, Kristof Schütt, Klaus-Robert Müller, and Sven Dähne. "Investigating the influence of noise and distractors on the interpretation of neural networks." *arXiv preprint arXiv:1611.07270*, 2016.

[KT00] Vijay R Konda and John N Tsitsiklis. "Actor-critic algorithms." In *Advances in neural information processing systems*, pp. 1008–1014, 2000.

[KWG17] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. "Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav)." *arXiv preprint arXiv:1711.11279*, 2017.

[LB15] Diego M López and Bernd Blobel. "mHealth in Low-and Middle-Income Countries: Status, Requirements and Strategies." In *pHealth*, pp. 79–87, 2015.

[LCL10] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. "A contextual-bandit approach to personalized news article recommendation." In *Proceedings of the 19th international conference on World wide web*, pp. 661–670. ACM, 2010.

[LEL18] Scott M Lundberg, Gabriel G Erion, and Su-In Lee. "Consistent Individualized Feature Attribution for Tree Ensembles." *arXiv preprint arXiv:1802.03888*, 2018.

[LGZ16] Alex M Lamb, Anirudh Goyal Alias Parth Goyal, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. "Professor forcing: A new algorithm for training recurrent networks." In *Advances In Neural Information Processing Systems*, pp. 4601–4609, 2016.

[LHH18] Xinrui Lyu, Matthias Hueser, Stephanie L Hyland, George Zerveas, and Gunnar Raetsch. "Improving Clinical Predictions through Unsupervised Time Series Representation Learning." *arXiv preprint arXiv:1812.00490*, 2018.

[LHL15] Yaojin Lin, Qinghua Hu, Jinghua Liu, and Jie Duan. "Multi-label feature selection based on max-dependency and min-redundancy." *Neurocomputing*, **168**:92–103, 2015.

[Lic13] M. Lichman. "UCI Machine Learning Repository.", 2013.

[LKW16] Zachary C Lipton, David Kale, and Randall Wetzel. "Directly modeling missing data in sequences with rnns: Improved classification of clinical time series." In *Machine Learning for Healthcare Conference*, pp. 253–270, 2016.

[LL17] Scott M Lundberg and Su-In Lee. "A unified approach to interpreting model predictions." In *Advances in Neural Information Processing Systems*, pp. 4765–4774, 2017.

[LM18] Yingzhen Li and Stephan Mandt. "Disentangled sequential autoencoder." *arXiv preprint arXiv:1803.02991*, 2018.

[LSL16] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. "Autoencoding beyond pixels using a learned similarity metric." In *International Conference on Machine Learning*, pp. 1558–1566, 2016.

[Mac10] A Mackinnon. "The use and reporting of multiple imputation in medical research–a review." *Journal of internal medicine*, **268**(6):586–593, 2010.

[MDG16] Scott McLachlan, Kudakwashe Dube, and Thomas Gallagher. "Using the caremap with health incidents statistics for generating the realistic synthetic electronic healthcare record." In *Healthcare Informatics (ICHI), 2016 IEEE International Conference on*, pp. 439–448. IEEE, 2016.

[Men94] Xiao-Li Meng. "Multiple-imputation inferences with uncongenial sources of input." *Statistical Science*, pp. 538–558, 1994.

[MH08] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." *Journal of machine learning research*, **9**(Nov):2579–2605, 2008.

[MO14] Mehdi Mirza and Simon Osindero. "Conditional generative adversarial nets." *arXiv preprint arXiv:1411.1784*, 2014.

[Mog16] Olof Mogren. "C-RNN-GAN: Continuous recurrent neural networks with adversarial training." *arXiv preprint arXiv:1611.09904*, 2016.

[MP10] Debashis Mondal and Donald B Percival. "Wavelet variance analysis for gappy time series." *Annals of the Institute of Statistical Mathematics*, **62**(5):943–966, 2010.

[MS04] Bradley Malin and Latanya Sweeney. "How (not) to protect genomic data privacy in a distributed network: using trail re-identification to evaluate and design anonymity protection systems." *Journal of biomedical informatics*, **37**(3):179–192, 2004.

[MSJ15] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. "Adversarial autoencoders." *arXiv preprint arXiv:1511.05644*, 2015.

[NS08] Arvind Narayanan and Vitaly Shmatikov. "Robust de-anonymization of large sparse datasets." In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pp. 111–125. IEEE, 2008.

[ODZ16] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. "WaveNet: A generative model for raw audio." In *SSW*, p. 125, 2016.

[PAB10] Ronald Carl Petersen, PS Aisen, LA Beckett, MC Donohue, AC Gamst, DJ Harvey, CR Jack, WJ Jagust, LM Shaw, AW Toga, et al. "Alzheimer's disease neuroimaging initiative (ADNI): clinical characterization." *Neurology*, **74**(3):201–209, 2010.

[PAE16] Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. "Semi-supervised knowledge transfer for deep learning from private training data." *arXiv preprint arXiv:1610.05755*, 2016.

[Pal07] Lyle J Palmer. "UK Biobank: bank on it." *The Lancet*, **369**(9578):1980–1982, 2007.

[PAM12] Stuart J Pocock, Cono A Ariti, John JV McMurray, Aldo Maggioni, Lars Køber, Iain B Squire, Karl Swedberg, Joanna Dobson, Katrina K Poppe, Gillian A Whalley, et al. "Predicting survival in heart failure: a risk score based on 39 372 patients from 30 studies." *European heart journal*, **34**(19):1404–1413, 2012.

[Pat02] Patricia A Patrician. "Multiple imputation for missing data." *Research in Nursing & Health*, **25**(1):76–84, 2002.

[PG02] Shahla Parveen and Phil Green. "Speech recognition with missing data using recurrent neural nets." In *Advances in Neural Information Processing Systems*, pp. 1189–1195, 2002.

[PLD05] Hanchuan Peng, Fuhui Long, and Chris Ding. "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy." *IEEE Transactions on pattern analysis and machine intelligence*, **27**(8):1226–1238, 2005.

[PS08] Jan Peters and Stefan Schaal. "Natural actor-critic." *Neurocomputing*, **71**(7-9):1180–1190, 2008.

[PS15] Archana Purwar and Sandeep Kumar Singh. "Hybrid prediction model with missing value imputation for medical data." *Expert Systems with Applications*, **42**(13):5621–5631, 2015.

[PSM18] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. "Scalable Private Learning with PATE." *arXiv preprint arXiv:1802.08908*, 2018.

[QSC17] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. "A dual-stage attention-based recurrent neural network for time series prediction." *arXiv preprint arXiv:1704.02971*, 2017.

[Qui86] J. Ross Quinlan. "Induction of decision trees." *Machine learning*, **1**(1):81–106, 1986.

[RFP19] Sandeep Reddy, John Fox, and Maulik P Purohit. "Artificial intelligence-enabled healthcare delivery." *Journal of the Royal Society of Medicine*, **112**(1):22–28, 2019.

[Ris01] Irina Rish. "An empirical study of the naive Bayes classifier." In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pp. 41–46. IBM, 2001.

[RPB18] Giorgia Ramponi, Pavlos Protopapas, Marco Brambilla, and Ryan Janssen. "T-CGAN: Conditional Generative Adversarial Network for Data Augmentation in Noisy Time Series with Irregular Sampling." *arXiv preprint arXiv:1811.08295*, 2018.

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should i trust you?: Explaining the predictions of any classifier." In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144. ACM, 2016.

[Rub04] Donald B Rubin. *Multiple imputation for nonresponse in surveys*, volume 81. John Wiley & Sons, 2004.

[SB11] Daniel J Stekhoven and Peter Bühlmann. "MissForest—non-parametric missing value imputation for mixed-type data." *Bioinformatics*, **28**(1):112–118, 2011.

[Sch08] Daniel Schunk. "A Markov chain Monte Carlo algorithm for multiple imputation in large surveys." *AStA Advances in Statistical Analysis*, **92**(1):101–114, 2008.

[SGK17] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. "Learning important features through propagating activation differences." In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3145–3153. JMLR. org, 2017.

[SHK14] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, **15**(1):1929–1958, 2014.

[SHR09] Fritz H Schröder, Jonas Hugosson, Monique J Roobol, Teuvo LJ Tammela, Stefano Ciatto, Vera Nelen, Maciej Kwiatkowski, Marcos Lujan, Hans Lilja, Marco Zappa, et al. "Screening and prostate-cancer mortality in a randomized European study." *New England Journal of Medicine*, **360**(13):1320–1328, 2009.

[Sim18] Luca Simonetto. "Generating spiking time series with Generative Adversarial Networks: an application on banking transactions." 2018.

[SK14] Erik Štrumbelj and Igor Kononenko. "Explaining prediction models and individual predictions with feature contributions." *Knowledge and information systems*, **41**(3):647–665, 2014.

[SMH11] Ilya Sutskever, James Martens, and Geoffrey E Hinton. "Generating text with recurrent neural networks." In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017–1024, 2011.

[SMS15] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. "Unsupervised learning of video representations using lstms." In *International conference on machine learning*, pp. 843–852, 2015.

[SSS16] Tobias Schnabel, Adith Swaminatan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. "Recommendations as treatments: debiasing learning and evolution." *ICML*, 2016.

[Sut88] Richard S Sutton. "Learning to predict by the methods of temporal differences." *Machine learning*, **3**(1):9–44, 1988.

[SVI16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the inception architecture for computer vision." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps." *arXiv preprint arXiv:1312.6034*, 2013.

[SWC09] Jonathan AC Sterne, Ian R White, John B Carlin, Michael Spratt, Patrick Royston, Michael G Kenward, Angela M Wood, and James R Carpenter. "Multiple imputation for missing data in epidemiological and clinical research: potential and pitfalls." *BMJ*, **338**:b2393, 2009.

[Swe97] Latanya Sweeney. "Weaving technology and policy together to maintain confidentiality." *The Journal of Law, Medicine & Ethics*, **25**(2-3):98–110, 1997.

[SWR10] Charles E Schroeder, Donald A Wilson, Thomas Radman, Helen Scharfman, and Peter Lakatos. "Dynamics of active sensing and perceptual selection." *Current opinion in neurobiology*, **20**(2):172–176, 2010.

[TB98] Volker Tresp and Thomas Briegel. "A solution for missing data in recurrent neural networks with an application to blood glucose prediction." *Advances in Neural Information Processing Systems*, pp. 971–977, 1998.

[TP18] David Tsay and Cam Patterson. "From Machine Learning to Artificial Intelligence Applications in Cardiac Care: Real-World Examples in Improving Imaging and Patient Access." *Circulation*, **138**(22):2569–2575, 2018.

[TPB00] Naftali Tishby, Fernando C Pereira, and William Bialek. "The information bottleneck method." *arXiv preprint physics/0004057*, 2000.

[UV11] Jonathan Ullman and Salil Vadhan. "PCPs and the hardness of generating private synthetic data." In *Theory of Cryptography Conference*, pp. 400–416. Springer, 2011.

[VLB08] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. "Extracting and composing robust features with denoising autoencoders." In *Proceedings of the 25th International conference on Machine learning*, pp. 1096–1103. ACM, 2008.

[VSP17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[WRG96] Milton C Weinstein, Louise B Russell, Marthe R Gold, Joanna E Siegel, et al. *Cost-effectiveness in health and medicine*. Oxford university press, 1996.

[WRW11] Ian R White, Patrick Royston, and Angela M Wood. "Multiple imputation using chained equations: issues and guidance for practice." *Statistics in medicine*, **30**(4):377–399, 2011.

[WZ89] Ronald J Williams and David Zipser. "A learning algorithm for continually running fully recurrent neural networks." *Neural computation*, **1**(2):270–280, 1989.

[XBK15] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. "Show, attend and tell: Neural image caption generation with visual attention." In *International conference on machine learning*, pp. 2048–2057, 2015.

[XLW18] Liyang Xie, Kaixiang Lin, Shu Wang, Fei Wang, and Jiayu Zhou. "Differentially Private Generative Adversarial Network." *arXiv preprint arXiv:1802.06739*, 2018.

[YAH16] Jinsung Yoon, Ahmed Alaa, Scott Hu, and Mihaela Schaar. "ForecastICU: a prognostic decision support system for timely prediction of intensive care unit admission." In *International Conference on Machine Learning*, pp. 1680–1689, 2016.

[YDS17] Jinsung Yoon, Camelia Davtyan, and Mihaela van der Schaar. "Discovery and clinical decision support for personalized healthcare." *IEEE journal of biomedical and health informatics*, **21**(4):1133–1145, 2017.

[YJS18] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. "GANITE: Estimation of Individualized Treatment Effects using Generative Adversarial Nets." In *International Conference on Learning Representations*, 2018.

[YJS19a] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. "INVASE: Instance-wise Variable Selection using Neural Networks." In *International Conference on Learning Representations*, 2019.

[YJS19b] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. "PATE-GAN: Generating Synthetic Data with Differential Privacy Guarantees." In *International Conference on Learning Representations*, 2019.

[YKR09] Shipeng Yu, Balaji Krishnapuram, Romer Rosales, and R. Bharat Rao. "Active Sensing." In *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, pp. 639–646, 2009.

[YRD16] Hsiang-Fu Yu, Hikhil Rao, and Inderjit S. Dhillon. "Temporal regularized matrix factorization for high-dimensional time series prediction." *NIPS*, 2016.

[YZB18] Jinsung Yoon, William R Zame, Amitava Banerjee, Martin Cadeiras, Ahmed M Alaa, and Mihaela van der Schaar. "Personalized survival predictions via Trees of Predictors: An application to cardiac transplantation." *PloS one*, **13**(3):e0194985, 2018.

[YZS18a] Jinsung Yoon, William R. Zame, and Mihaela van der Schaar. "Deep Sensing: Active Sensing using Multi-directional Recurrent Neural Networks." In *International Conference on Learning Representations*, 2018.

[YZS18b] Jinsung Yoon, William R Zame, and Mihaela van der Schaar. "ToPs: Ensemble Learning With Trees of Predictors." *IEEE Transactions on Signal Processing*, **66**(8):2141–2152, 2018.

[ZGC16] Yizhe Zhang, Zhe Gan, and Lawrence Carin. "Generating text via adversarial training." In *NIPS workshop on Adversarial Training*, volume 21, 2016.

[ZKK18] Chi Zhang, Sanmukh R Kuppannagari, Rajgopal Kannan, and Viktor K Prasanna. "Generative Adversarial Network for Synthetic Time Series Data Generation in Smart Grids." In *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pp. 1–6. IEEE, 2018.

[ZKZ17] Jake Zhao, Yoon Kim, Kelly Zhang, Alexander M Rush, and Yann LeCun. "Adversarially regularized autoencoders." *arXiv preprint arXiv:1706.04223*, 2017.