Pipelines for Deep Contextual Patient-Level Clinical Outcome Prediction

by

Rohan Kodialam

S.B., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2020

Author .....
Department of Electrical Engineering and Computer Science
July 10, 2020

Certified by .....
David Sontag
Associate Professor
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Pipelines for Deep Contextual Patient-Level Clinical Outcome Prediction

by

Rohan Kodialam

Submitted to the Department of Electrical Engineering and Computer Science
on July 10, 2020, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Longitudinal health data provides a uniquely detailed view into the evolution of patient health over time. We develop pipelines to efficiently work with this kind of data in its rawest form, enabling the development of new state-of-the-art end-to-end machine learning approaches. While healthcare providers are increasingly using learned methods to predict and understand long-term patient outcomes in order to make meaningful interventions, deep learning models often struggle to match performance of shallow linear models in predicting these outcomes, making it difficult to leverage such techniques in practice. Motivated by the task of clinical prediction from longitudinal health data, we present a new technique called *reverse distillation* which pre-trains deep models by using high-performing linear models for initialization. We make use of the longitudinal structure of our dataset to develop Self Attention with Reverse Distillation, or SARD, an architecture that utilizes a combination of contextual embedding, temporal embedding and self-attention mechanisms and most critically is trained via reverse distillation. SARD outperforms state-of-the-art methods on multiple clinical prediction outcomes, with ablation studies revealing that reverse distillation is the primary driver of these improvements.

Thesis Supervisor: David Sontag
Title: Associate Professor

Acknowledgments

The work presented in this thesis would not have been possible without the collaboration and support of many people, to whom I am deeply thankful. Foremost amongst them is my advisor, David Sontag, who has helped me become a better researcher. His uncanny ability to always ask the right questions has led to incredibly insightful conclusions instrumental in the development of my work, and I am grateful that some of his curiosity, sharp inquisitiveness, and tireless pursuit of excellence have been imparted to me. The impact his unparalleled knowledge of the field has had on my own learning cannot be overstated.

I would also like to thank the rest of the Clinical Machine Learning group, particularly Rebecca Boiarsky, who helped guide me through this exciting year of research. Our group's endless ideation and discussion, whether in-person at MIT or over video calls during the current pandemic, was a source of inspiration and enjoyment. Along with our team at MIT, I also had the pleasure of working with Aaron Smith-McLallen and the data science group at Independence Blue Cross – their expertise, data, and support of my research were invaluable to me.

Thanks are also due to the long litany of faculty and fellow students who have educated me over my five years at MIT. In particular, Alexander Rakhlin, my previous UROP advisor, helped me develop the skill-set needed to perform research in machine learning. I am very grateful for his guidance and support. I also thank my undergraduate academic advisors Pablo Jarillo-Herrero from the Physics Department and Suvrit Sra from the EECS Department, who were endlessly supportive of my constant reshuffling of classes and petitioning of graduation requirements.

Finally, I express my gratitude to my friends and family, who (whether voluntarily or not) continue to act as a sounding-board for every half-baked idea that crosses my mind. Their unwavering support has made my time at MIT an absolute pleasure.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem and Motivation

Clinical prediction is critical in providing preventative, prophylactic and palliative care. At a fundamental level, the process of medical decision-making is a question of estimating risk probabilities in an uncertain setting [3]. Clinicians attempt to discover a patient's current state through examination and testing, and based on this information decide what steps to take to mitigate the potential for adverse outcomes in the future.

While traditionally the prediction of outcomes has followed from expert intuition and experience, the advent of electronic health records (EHRs) and other digital stores of structured medical data has allowed for a proliferation of data-driven approaches to this task [7]. Several rule-based approaches to determining how a patient will fare in the future are currently in wide use by physicians, albeit in a somewhat auxiliary role in which the recommendations of rule-based models are combined heuristically with clinical intuitions to make final decisions [8]. Doctors find many reasons to mistrust such models – for example, rule-based models may not factor in a key feature that a clinician finds significant [8]. Despite this, medical research has found that prediction models are often more accurate than clinicians. For example, prior work has found that when predicting patient mortality, simple algorithmic predictors outperform predictions inferred from the decisions of human experts. [4].

Meta-analyses of clinical prediction strategies broadly split these tasks into three categories:

- Diagnosis: determining a patient's current state by estimating probabilities of whether or not a condition affects a patient based on current observations and patient history.
- Prescription: determining how to treat a patient to mitigate future risk, by predicting a patient's future health conditional on the administration of different treatments and the patient's current health.
- Prognosis: determining a patient's future state, based solely on their present and historical health statuses.

While diagnosis tasks are dominated by a combination of modelling and live clinical intuition, and prescription tasks run the risk of drawing incorrect causal conclusions when run solely on retrospectively collected data, prognosis remains an extremely promising field for innovation. With access to sufficiently detailed medical records, prognoses can be performed at regular intervals to detect potential risks well in advance of their manifestation. If predictions are sufficiently accurate, appropriate interventions can then be taken to mitigate these future risks.

Thus, in this thesis, we explore algorithmic solutions to the challenge of using the longitudinal medical history of a patient to make predictions of the patient's future outcomes. Our novel deep learning approaches, combined with engineering innovations in the pipeline used to extract information from standardized databases of longitudinal medical data, allow us to significantly outperform the prior state-of-the-art in clinical prediction. These increases in performance can then be translated into more accurate and meaningful preventative interventions.

1.2 Thesis Roadmap

In this section, we outline the core components of the thesis, and indicate how these components depend on each other to form a coherent system for learning to predict

clinical outcomes. In Chapter 2, we describe the general structure of the longitudinal EHR data that our models ingest, and specifically discuss the layout of the Observational Medical Outcomes Partnership (OMOP) Common Data Model (CDM) [23], which represents a standardized way to represent longitudinal health data. Chapter 3 discusses Prediction Library, an open-sourced pipeline we developed in order to set up clinical prediction problems and to move the appropriate raw data from the OMOP CDM databases described in Chapter 2 into data structures designed to make downstream modelling efficient. Next, in Chapter 4 we develop simple yet performant linear models that use features derived from the data representations explained in Chapter 3. These models serve as baselines for our more advanced machine learning algorithm, which we describe in Chapter 5. This algorithm, which we name SARD, uses a unique combination of data embedding techniques and a self-attention mechanism to ingest longitudinal EHR data. In addition, SARD is trained using *reverse distillation*, a process we develop in Chapter 6 which allows us to use a high-performing linear model, as constructed in Chapter 4, to initialize and regularize SARD. We finally evaluate the performance of our algorithms on several examples of real clinical prediction tasks with applications to preventative and palliative care in Chapter 7, and establish significant performance gains versus prior results. In addition to confirming performance increases, in Chapter 7 we further introspect into the SARD model, identifying that reverse distillation is the main driver of gain and developing a method to achieve a degree of patient-level interpretability of predictions.

Chapter 2

A Summary of Longitudinal Health Data

We discuss the form and structure of longitudinal health data in electronic health records (EHRs). Our particular focus is on *claims data*, which are the records maintained by insurers of their members' interactions with the healthcare system. Such data provides an invaluable and detailed view of patient health over time, as well as the decisions taken by clinicians regarding patient care.

2.1 The Structure of Longitudinal EHR data

2.1.1 Patients

We consider the data associated with a single patient. While the majority of EHR data is longitudinal in nature, there are relatively fixed metadata that can be associated with each individual in the system. Such information may include a person's location of residence over time, date of birth, gender and ethnicity. Additionally, in insurance claims data, it is often the case that one can observe the type of insurance plan a patient is on, which can then act as a proxy for socioeconomic variables.

These variables clearly can help contextualize the health of a patient, and as such are useful from a modelling perspective. They also come into play during the

process of evaluating a predictions, as we would want to closely study differences in the performance of any predictive algorithm across different demographic classes in order to identify and potentially correct for any biases.

2.1.2 Visits

Longitudinal EHR datasets store patients' medical history in an ordered and hierarchical way. At the level of an individual patient, the primary top-level unit into which data is grouped is the *visit*, which represents a single continuous interaction of the patient with the healthcare system. A visit may represent an inpatient hospital stay, outpatient treatment, a consultation with a physician, and other miscellaneous interactions.

A set of metadata is associated with each visit, allowing it to be understood in context. This data includes the type of visit (e.g. inpatient versus outpatient), the start and end dates of the visit, the type of medical facility where the visit occurred, the specific facility where the visit occurred, and the identity and medical specialty of the physician who supervised the visit.

2.1.3 Codes

To represent the medical events that occur during a visit, a large fixed set of *codes* are used, with each code representing a distinct and specific medical concept. These codes can be drawn from a variety of rich clinical vocabularies including the 10th revision of the International Statistical Classification of Diseases (ICD-10), the Healthcare Common Procedure Coding System (HCPCS) and the Systematized Nomenclature of Medicine (SNOMED).

Codes are designed to be as specific as possible, and it is therefore possible to define a hierarchy over codes. By introducing higher-level codes to represent groupings of medical concepts, we can identify specific concepts as subsets of more general medical ideas. This kind of meaningful grouping allows for easier interpretation and ingestion of the data encapsulated in the codes.

A variety of data are represented by codes, of which the following are of primary importance:

- The diagnoses made by a clinician regarding a patient's condition, and any accompanying observations of symptoms.
- Medical procedures that were carried out on a patient.
- Medications prescribed to a patient, or used while care was being administered.
- Medical devices used on or assigned to a patient, such as catheters and stents.
- Laboratory tests conducted on patients, along with categorical indications of their results

In addition to these types of data, different EHR systems may include additional information based on availability, ease of data collection, and the ultimate use cases for the data

Associated with each code is additional metadata, including the time when the code was assigned and the physician who assigned the code. In cases where a code represents a measurement or another concept associated with a numerical value, this numerical value may also appear as a type of code-level metadata.

2.2 The OMOP CDM

In the previous section we describe at a high level the components of a longitudinal EHR data store. It is quite clear that these specifications do not uniquely determine a database structure. In addition, as different institutions have different uses for clinical data, it is natural that over time a variety of EHR systems have emerged, with varying levels of compatibility.

In order to make research and insights more transferable between institutions, the Observational Medical Outcomes Partnership Common Data Model (OMOP CDM) was designed to act as a way to define relational databases to contain all of the data commonly found in EHRs in a consistent manner. As researchers generally hope

that their work can be generalized across institutions, using the OMOP CDM is a useful choice for the development of applications that ingest EHR data, and our work likewise uses an OMOP-standard relational database as its starting point. In this section, we discuss in detail the structure of the OMOP CDM.

2.2.1 Standardized Clinical Data Tables

Standardized Clinical Data Tables record information about patients, their visits, and the interactions with the health system that occur during each visit. Figure 2-1 displays the structure of each of the standardized tables. Of particular importance to our applications are:

- The `person` table, which contains an entry for each person in the data set, and provides relevant demographic information such as age and gender.
- The `visit_occurrence` table, which lists when visits occur for each patient, the type of visit, and the clinician in charge of the visit.
- The `condition_occurrence`, `drug_exposure`, `device_exposure` and `procedure_occurrence` tables, which list when a condition is observed, a drug is administer, a medical device used, or a procedure undertaken on a patient, respectively. These tables, along with the other tables marked as descendants of the `visit_occurrence` table in Figure 2-1, contain patient-level longitudinal records of what medical events occurred during each visit.

In order to represent abstract medical concepts, such as the diagnosis of a specific disease or a specific type of clinician specialty in the Standardized Clinical Data Tables, the OMOP CDM relies of a robust, hierarchical vocabulary, which we describe in the next section.

2.2.2 Standardized Vocabularies

The OMOP Standardized Vocabularies unify many medical vocabularies and their respective codes into a single common format. The design of the Standardized Vo-

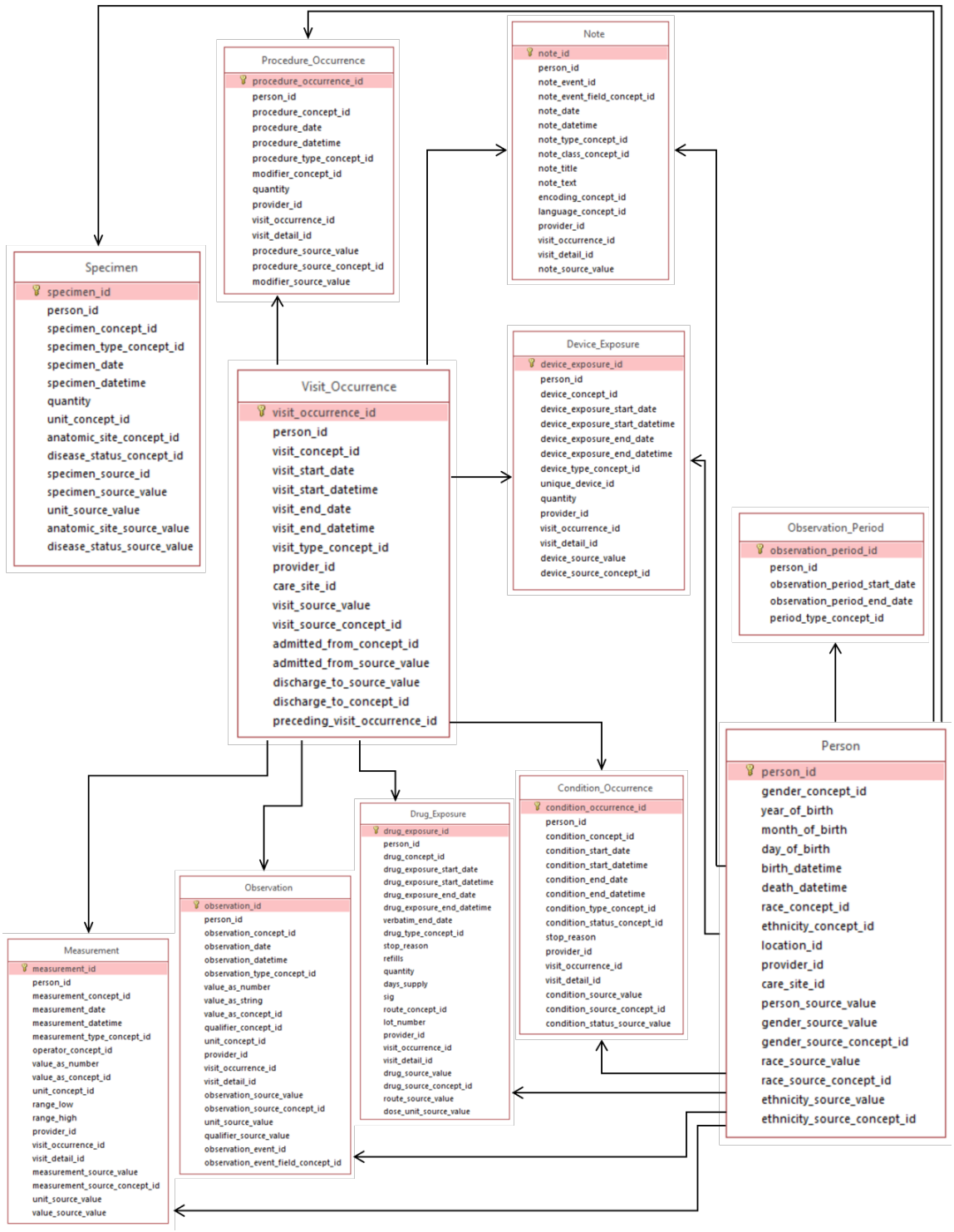

Figure 2-1: OMOP Standardized Clinical Data Tables, sourced from the OMOP Wiki [1]

cabularies ensures that several desiderata are satisfied [1]:

- Only a single standardized vocabulary is used, allowing researches to learn only one convention for the representation and manipulation of clinical concepts

- Every concept in the standardized vocabulary is unique, meaning that synonymous terms from different input vocabularies are merged to reflect their common meaning. A critical example of the benefit of this design decision is the manner in which algorithms running on EHR were affected by the shift between the ICD-9 and ICD-10 vocabularies [25]. Systems that were reliant on ICD-9 codes would suddenly find that no data was available as doctors began to use newer coding practices, whereas using the OMOP CDM, one would find that new ICD-10 concepts and equivalent ICD-9 concepts would be mapped to the same code in the standardized vocabulary, allowing for a more seamless transition. To facilitate this, the OMOP CDM maintains the `concept_synonym` table, which connects the terms in various vocabularies to a common standardized concept.

- Coverage of medical concepts is comprehensive, and any event relevant to the healthcare experience should be captured by an OMOP concept. Updates are regularly delivered to reflect new types of events that may not have been relevant in the past.

The standardized vocabulary is split into *domains*, several of which are of immediate interest in terms of data collection for predictive modelling. Thus, we next discuss the most important of these domains.

- The *condition* domain's codes represent the reporting of a disease or symptom for a patient. The set of diseases and symptoms is, as per the principles of the standardized vocabularies, unique and exhaustive. It builds on the SNOMED vocabulary [47], and is augmented with various other commonly used terminologies. Furthermore, this domain is hierarchical, and it uses a combination of SNOMED and MedDRA [9] to represent a wide range of granularities, from closely related conditions to the set of all conditions affecting a specific organ

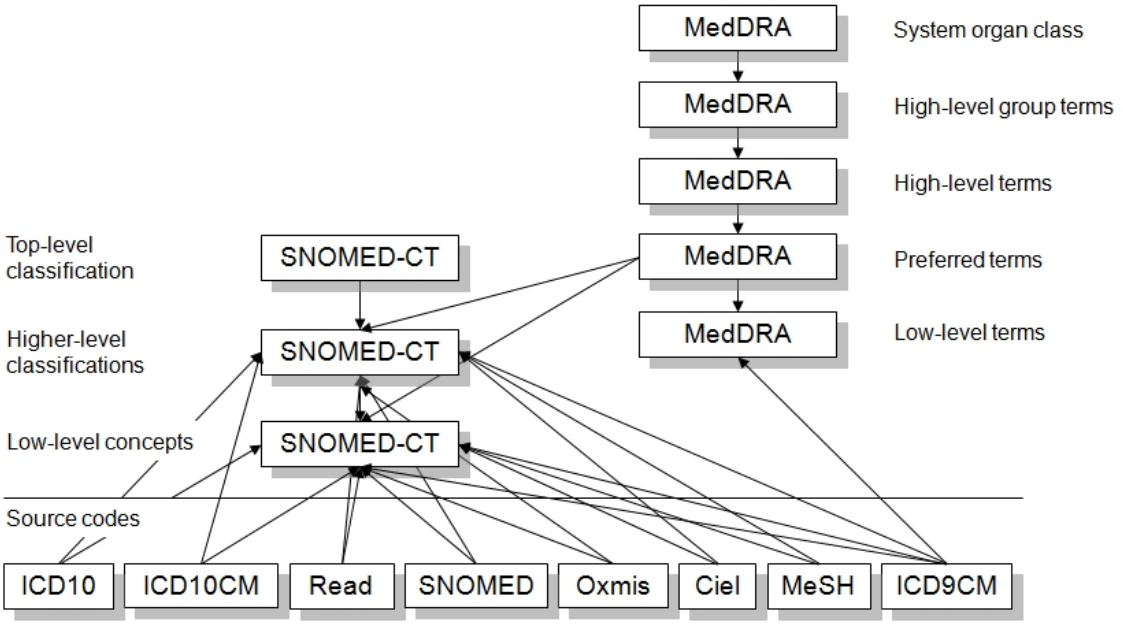

Figure 2-2: OMOP Condition Domain Diagram, sourced from the OMOP Wiki [1]

system. We display the structure of the condition domain, and its constituent vocabularies in Figure 2-2

To help clarify the hierarchical nature of OMOP codes, we display a part of the hierarchical tree for the condition domain in Figure 2-3.

- The *drug* domain, whose codes represent the utilization of any biochemical substance as part of patient care, and the closely related *device* domain, which represents any instruments, implants, reagents that are used as part of care but do not have an effect which is chemical in nature. The drug domain draws from a large set of vocabularies, as seen in Figure 2-4, in order to identify and relate unique pharmacological ingredients, the drugs derived from combinations of these ingredients, and the brand-name products that may be used in medical practice or prescribed to a patient.

- The *procedure* domain, which contains codes for any medical procedure carried out by a clinician on a patient. This domain largely relies on the HCPCS vocabulary and associated hierarchy of procedures [19], but also contains mappings for codes from the ICD-10-PCS vocabulary as well [6].

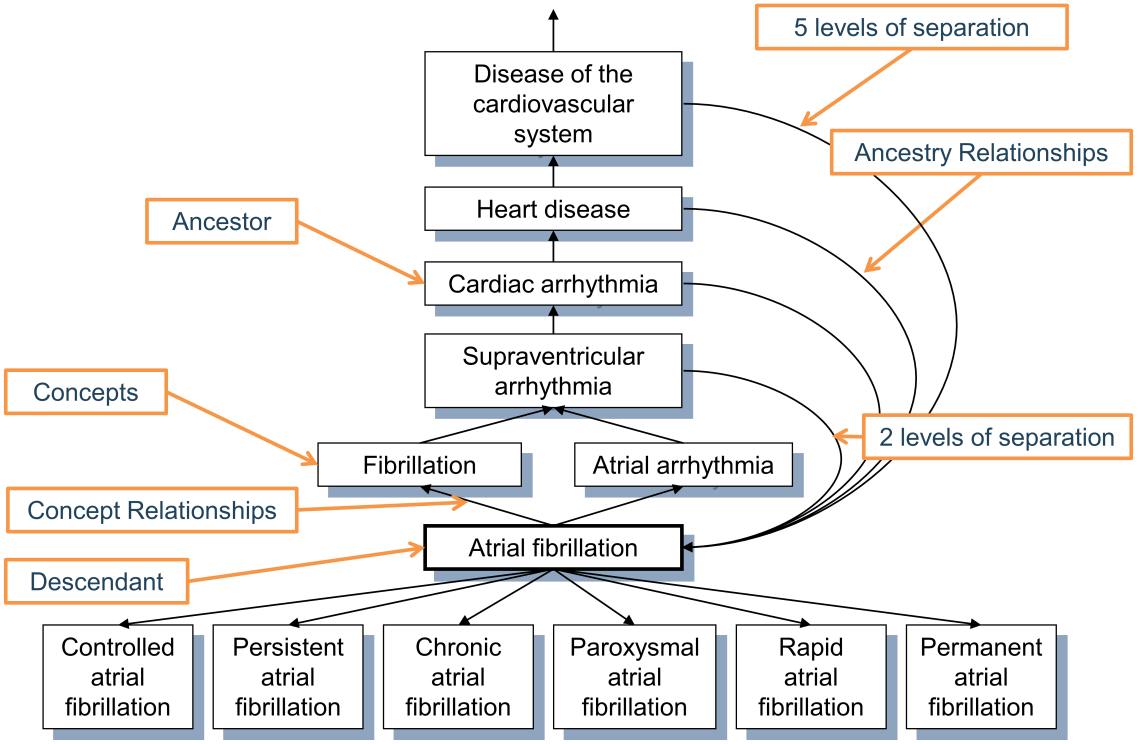

Figure 2-3: OMOP Condition Heirarchy Example, sourced from the OMOP Wiki [1]

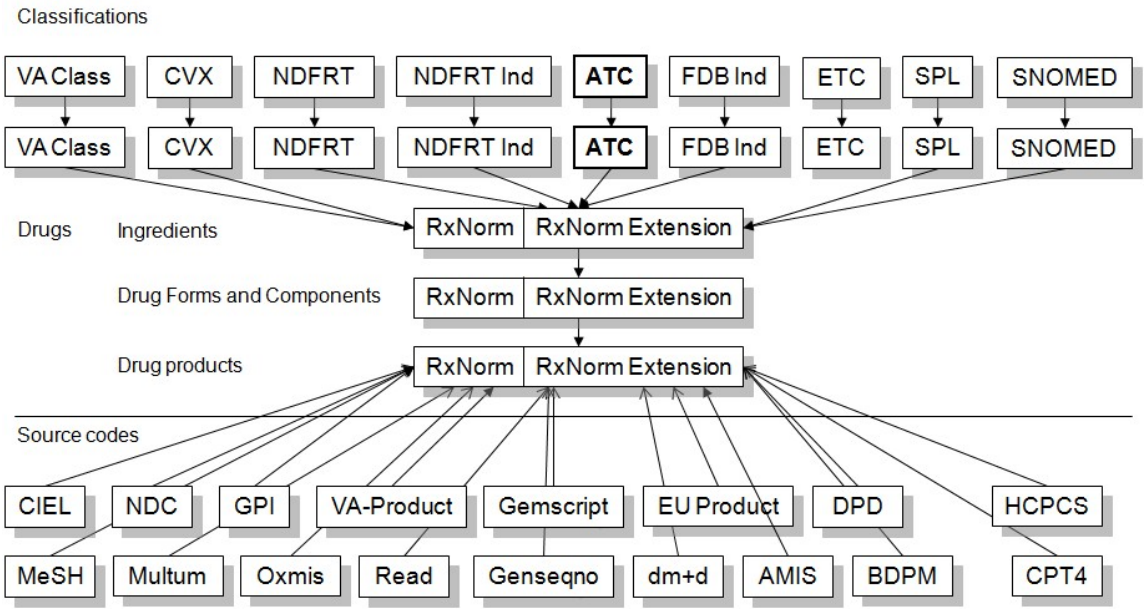

Figure 2-4: OMOP Drug Domain Diagram, sourced from the OMOP Wiki [1]

- The *provider specialty* domain, which contains codes corresponding to different clinician specialties. These codes can be linked to each visit, identifying the type of doctor who supervised care. Unlike other previously discussed domains, the *provider specialty* domain is not hierarchical in nature and is simply represented as a list of specialty types. Specialties are based on the SNOMED vocabulary standard [47].

Concepts from the standard vocabulary are reflected in the concepts table of the OMOP CDM. The set of domains discussed reflects the concepts used in our downstream applications, but is by no means exhaustive. We refer the reader to OMOP's own documentation [1] for further details on other types of information encapsulated into an OMOP CDM, such as administrative details of medical care, demographic data, and records of qualitative and quantitative measurements taken during the course of care.

Chapter 3

Prediction Library Data Pipeline

In order to allow algorithms to efficiently operate on EHR data stored in an OMOP CDM relational database, it is necessary to represent information in a manner that is easy to computationally ingest. The primary challenge to accomplishing this goal is the very large space of potential medical timelines that each patient can have. Indeed, a patient can be assigned an arbitrary subset of codes on each day for which clinical history is available.

In order to store and manipulate subsets of this data efficiently for the purpose of predictive modelling, we developed Prediction Library, an open-sourced set of tools in the Python language that allow for very general data collection and organization on top of an OMOP CDM database. Prior work in this space has focused on similar applications in the R language, with similar capacity for general definitions of prediction tasks but less flexibility in terms of arbitrary manipulation of large amounts of longitudinal health data [46].

3.1 Defining a Clinical Prediction Task

In order to build Prediction Library's pipelines, we first formally define the components of a clinical prediction task. A prediction task has four primary components:

- A **Cohort** of patients of interest, who are selected based on a combination of medical and data-driven criteria. For example, we may only be interested in

patients of a certain age group or gender, or only those patients for whom we have sufficient claims data. More generally, the inclusion criteria for a cohort can be expressed as a set of rules at a per-patient level.

- An **Outcome**, defined per-patient as a binary indicator of whether a specified medical event occurs within a given window. We generically denote this time frame by the interval $[T_B, T_C]$. Note that T_B can possibly be later than the time of prediction T_A, indicating that there is a gap of length $T_B - T_A$ between the last data points available and the outcome determination window. This gap is necessary since predicting an imminent medical event may not be useful in certain use cases. For example, if the goal of a predictive model is to determine which patients require clinical intervention, the prediction of an imminent medical event may not be useful as it is too late to take meaningful preventative actions.

- A **Prediction Time**, which we denote by T_A, representing when we make a prediction. All data timestamped after T_A must be censored from predictive algorithms, with the singular exception of the measurement of the outcome of interest, in order to serve as training and evaluation data for models. In practice, the prediction time is only useful for defining prediction problems for model building – when deployed in a clinical setting T_A will be the time at which the model is actually run and thus censoring of information is not relevant.

- **Predictive Features** are some subset of the available non-censored EHR data that are used as inputs to a predictive algorithm. Similar to the definition of the cohort, we seek to define the covariates available to such an algorithm dynamically on a per-task basis.

We now detail the process of defining a specific predictive task. The first and easiest step is to select a prediction time T_A. Next, from the set of all patients $\mathcal{P}$, we define a rule χ that maps each patient to a Boolean value indicating if the patient is to be included in the cohort. By applying this rule, we can collect the

cohort $\chi(\mathcal{P}) = \{p \in \mathcal{P} | \chi(p)\}$. In tandem with χ, we define the binary function $\Omega : \mathcal{P} \to \{0, 1\}$ defining an outcome for each patient. Ultimately, using these two functions we can create list of patients in $\chi(\mathcal{P})$ along with their clinical outcomes.

With a cohort and outcomes defined, it remains to collect the relevant features which will be used for prediction. These features are arbitrary transformations of the data held in an OMOP CDM. In particular, we define a set of features generation functions $c_1, c_2, \cdots c_n$, with each function c_i assigning each patient in $\chi(\mathcal{P})$ a time-series of binary indicators indicating whether or not the feature represented by c_i is applicable to the patient at each possible timestep, as measured in days. Generalizing the language of the OMOP CDM, we notate that feature generation function c_i indicates that its feature is relevant to patient p on day t as p having code c_i assigned at t, and as such we can represent the entire collected dataset as a list of tuples (p, c, t) corresponding to the statements that patient p was assigned code c at time t, with t

3.1.1 Representing Data Efficiently

As described in Section 3.1, we are able to collect a list of features of interest. Each element of this list is a tuple of the form (p, c, t) corresponding to the statement that patient p was assigned code c at time t, with t discretized to a daily level. We discuss the types of operations we hope to be able to conduct on this data in Section 3.1.1.1, along with naive algorithms to implement these operations. These algorithms have long runtimes, and as such we develop a different representation of the data that allows for significantly faster operations at the cost of a larger memory footprint in Section 3.1.1.2. Finally, we develop an intermediate representation that achieves acceptably fast runtimes with a small memory footprint in Section 3.1.1.3, which we implement in Prediction Library.

3.1.1.1 Data Representation Goals and Naive Implementation

In order to discuss the desired operations on the data, we first establish some notation. We denote the entire set of such tuples by $\mathcal{R}$, and further denote the set of unique

patients, codes and times in the dataset by $\mathcal{P}, \mathcal{C}, \mathcal{T}$ respectively. We further define the quantity $A = \max(|\mathcal{P}|, |\mathcal{C}|, |\mathcal{T}|)$ to represent the size of the largest of the these three sets. Note that asymptotically $|\mathcal{R}| \in O(|\mathcal{P}||\mathcal{C}||\mathcal{T}|) \in O(A^3)$, although in practice this bound is quite weak since the vast majority of codes will not apply to the a given patient at any given time. Our data is thus effectively stored as a long list of tuples of the form (p, c, t), which while efficient in terms of memory is not very amenable to transformation in an efficient manner. There are various ways in which one could manipulate these features for purposes such as further feature engineering. To support these manipulations, we seek efficient implementations of the following operations:

- Filter out any set of times, patients, or codes from the data based on arbitrary rules. As an example, we may want to collect all data occurring in a certain calendar year.

  In the most general sense, a user should be able to define filtering criteria as a tuple $f = (f_p, f_c, f_t)$ of Boolean functions for patients, codes and times respectively and we would return the set

  $$f(\mathcal{R}) = \{(p, c, t) \in \mathcal{R} | f_p(p) \wedge f_c(c) \wedge f_t(t)\}. \tag{3.1}$$

  This kind of filtering would require iterating through each tuple and thus take time $O(|\mathcal{R}|)$ Returning to our example, we could define an interval I equivalent to the year in question, in which case we would set $f_t(t) = [[t \in I]]$ and f_p, f_c to always be true.

- Aggregate data across an arbitrary dimension, which can either be the patient dimension $\mathcal{P}$, code dimension $\mathcal{C}$, or time dimension $\mathcal{T}$. We seek to optimize the aggregation operation because it is a critical tool both for crafting features and for introspecting into the properties of the available data. As a motivating example, we can try to find the distribution of codes over time. Therefore, we would want to sum up our data along the patient axis, leaving a two-dimensional

tensor indexed by time and code.

In general, the aggregation is done using an arbitrary *aggregation function* $g(a, x)$, where a represents the value to be incorporated into the aggregation and x holds the aggregated value so far. The aggregation function is constrained to be permutation invariant – that is, regardless of the order in which elements of the dimension in question are fed into the aggregation, the end result will be the same.

For example, consider the example of aggregating over patients. In performing a single iteration of the aggregation, we would use a value $p \in \mathcal{P}$ to collect all data tuples $\{(p', c, t) \in \mathcal{R} | p' = p\}$ which pertain to patient p and use these alongside the aggregated value so far to update the aggregated value. To represent this, we introduce the *primed aggregation function* g' defined such that

$$g(p, x) = g'(\{(p', c, t) \in \mathcal{R} | p' = p\}, x). \tag{3.2}$$

Then, starting from an initial value $x_\varnothing$, we would recursively aggregate by computing in a loop over each $p \in \mathcal{P}$ the function

$$x_{S \cup \{p\}} = g\left(p, x_S\right) = g'\left(\{(p', c, t) \in \mathcal{R} | p' = p\}, x_S\right) \tag{3.3}$$

where S represents the set of elements of $\mathcal{P}$ that have already been aggregated over. The final aggregation would be the value of $x_\mathcal{P}$, which would incorporate all patients. The process for aggregating over times and codes follows naturally.

Then, naively, this manner of aggregating along an dimension would first require finding every unique element of that dimension, iterating through each of these unique elements, finding the set of tuples containing the unique element, and finally actually performing the aggregation itself. Denoting the runtime of the aggregation function by T_g, naive aggregation would cost an asymptotic runtime of $O(T_g A|\mathcal{R}|)$, making it infeasible in practice.

To make this logic more concrete, we now return to the use case in which we

seek to measure the distributions of codes assigned at each time, which may be useful for analyses such as the quantification of dataset shift. Then, we would want to aggregate over individual patients using the matrix-valued aggregation function $g : \mathcal{P} \times \mathbb{N}^{|\mathcal{C}|\times|\mathcal{T}|} \to \mathbb{N}^{|\mathcal{C}|\times|\mathcal{T}|}$ defined by

$$g(p, x) = g'(\{(p', c, t) \in \mathcal{R} | p' = p\}, x) = x + m(p) \tag{3.4}$$

where the elements of the $|\mathcal{C}| \times |\mathcal{T}|$-dimensional matrix $m(p)$ are given by

$$m_{c,t}(p) = \begin{cases} 1 & (p, c, t) \in \{(p', c, t) \in \mathcal{R} | p' = p\} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

Starting from an initial value of the zero matrix $x_\varnothing = \mathbf{0}^{|\mathcal{C}|\times|\mathcal{T}|}$, this aggregation would then result in a matrix $x_\mathcal{P}$ whose row indices correspond to unique times, with the vector formed by each row representing an un-normalized empirical distributions of the codes that occurred at the time corresponding to the row.

3.1.1.2 Fast Implementation without Memory Constraints

We next develop significantly faster implementations of the filtering and aggregation operations. To do so, we first consider the case where memory usage is neglected – this leads to a data structure that can perform both operations very quickly, but has a large memory footprint.

In the memory-unconstrained scenario, we could store the list of tuples $\mathcal{R}$ in a large tensor $M(\mathcal{R}) \in \{0, 1\}^{|\mathcal{P}|\times|\mathcal{C}|\times|\mathcal{T}|}$ which we equip with three invertible functions h_p, h_c, h_t which map from the integer ranges $[|\mathcal{P}|], [|\mathcal{C}|], [|\mathcal{T}|]$ to the sets $\mathcal{P}, \mathcal{C}, \mathcal{T}$ respectively. Note that we denote the set of integers $\{1, 2, \cdots N\}$ as $[N]$. These functions and their inverses are each expressed as hash tables, allowing for $O(1)$ conversion in either direction between a numerical index in the matrix $M(\mathcal{R})$ and the corresponding patient, code or time. Using these definitions, we can define the elements of $M(\mathcal{R})$

by

$$M_{i,j,k}(\mathcal{R}) = \begin{cases} 1 & (h_p(i), h_c(j), h_t(k)) \in \mathcal{R} \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

Note that just as a matrix can be stored in row-major or column-major order, the three-dimensional sparse tensor can be stored in any of 6 axis-wise orders, with each order allowing for immediate slicing along one axis. If we store a copy of the tensor in each of these order, we can slice out any two-dimensional tensor slice in constant time.

Using this data structure, we can perform filtering according to the filtering criteria tuple $f = (f_p, f_c, f_t)$ by iterating through the unique patient, code, and time indices. When iterating through the patient indices, at index $i \in [|\mathcal{P}|]$ we first find the patient $p = h_p(i)$ and then check whether this patient is to be included in the filter by calculating $f_p(h_p(i))$. The process for checking inclusion of code and time indices follows naturally. In this way, assuming that the calculation of f_p, f_c, f_t can be done in constant time, the appropriate tensor elements can be found in time $O(|\mathcal{P}| + |\mathcal{C}| + |\mathcal{T}|) \in O(A)$.

We likewise see gains in terms of aggregation. To aggregate along a dimension, we merely iterate through each index of that axis. As an example, consider aggregation along the patient dimension – for each index $i \in [|\mathcal{P}|]$ the two-dimensional tensor $M_{i,\cdot,\cdot}$ is immediately accessible, and this can then be fed into the aggregation function. Given that the aggregation is permutation invariant, we can simply iterate through $[|\mathcal{P}|]$ instead of $\mathcal{P}$ itself and still go through every patient. Thus, the runtime of aggregation is decreased to $O(T_g A)$, where we recall that T_g is the runtime of the aggregation function.

3.1.1.3 Addressing both Memory and Runtime Constraints with Sparse Tensors

Unfortunately, it is often infeasible in terms of memory to store a single copy of $M(\mathcal{R})$, let alone 6 copies. Thus, we turn to an intermediate solution – sparse three

dimensional tensors. We represent these tensors by a $3 \times |\mathcal{R}|$ matrix $D(\mathcal{R})$ of natural numbers, with each column corresponding to a tuple in $\mathcal{R}$. More specifically, we define an arbitrary ordering over $\mathcal{R}$ such that for each $i \in [|\mathcal{R}|]$, we have a tuple $r_i = (p_i, c_i, t_i)$ corresponding to a unique element of $\mathcal{R}$. We then set the columns of $D(\mathcal{R})$ to

$$D_{i,\cdot}(\mathcal{R}) = \begin{bmatrix} h_p^{-1}(p_i) \\ h_c^{-1}(c_i) \\ h_t^{-1}(t_i) \end{bmatrix} \tag{3.7}$$

where the functions h_p, h_c, h_t are defined in the same way as in the case of a dense tensor above.

Efficient filtering and aggregation algorithms on this data structure achieve intermediate runtimes while conserving memory, and as such are useful practically:

- In order to filter according to the filtering criteria tuple $f = (f_p, f_c, f_t)$, we can first filter patients using f_p, then codes with f_c and finally times with f_t. For patients, we filter by iterating through each column, extracting the patient index i from the first element of the column, finding the corresponding patient $h_p^{-1}(i)$ and including the column if $f_p(h_p^{-1}(i))$ holds true. The procedure for filtering in codes and times follows naturally. Similar to the naive case, this incurs a $O(|\mathcal{R}|)$ runtime, which is acceptable in general due to the sparsity of the matrix.

- To aggregate, we can make use of the fact that D is sorted. First, we re-sort the columns of D using the aggregation dimension as the key in time $O(|\mathcal{R}| \log |\mathcal{R}|)$. Then, the relevant blocks of data of the form $\{(p, c, t') \in \mathcal{R} | t' = t\}$, $\{(p, c', t) \in \mathcal{R} | c' = c\}$, or $\{(p', c, t) \in \mathcal{R} | p' = p\}$ will simply be contiguous blocks of the matrix, resulting in a total runtime of $O(|\mathcal{R}| \log |\mathcal{R}| + T_g |\mathcal{R}|)$.

In this manner, we can achieve reasonable runtimes for both filtering and aggregation, while maintaining a small memory footprint. Similar methods for manipulating sparse data have historically been used in database applications [36]. In practical terms, within Prediction Library's code base we use the PyData-Sparse library [2], which

Data Structure	Filtering Runtime	Aggregation Runtime	Memory Footprint
List of Data	$O(\|\mathcal{R}\|)$	$O(T_g A\|\mathcal{R}\|)$	$O(\|\mathcal{R}\|)$
3-d Dense Tensor	$O(A)$	$O(T_g A)$	$O(A^3)$
3-d Sparse Tensor	$O(\|\mathcal{R}\|)$	$O(T_g\|\mathcal{R}\| + \|\mathcal{R}\| \log \|\mathcal{R}\|)$	$O(\|\mathcal{R}\|)$

Table 3.1: Memory and Runtime Tradeoffs when Storing Longitudinal Health Data

implements many of the sparse tensor operations we described above. We augment this library's sparse tensor logic with the mappings h_p, h_c, h_t and their inverses, to allow for clinically meaningful filtering and aggregation. We summarize the tradeoff between runtime and memory for the three different data structures discussed in Table 3.4.

3.2 Prediction Library Pipeline

Having discussed how a clinical prediction task is to be formally specified, and likewise how we can efficiently represent data, we finally turn to the engineering question of building an infrastructure to support these definition and data-collection processes. Our primary strategy to do so is to allow users to define all functions discussed above in terms of SQL scripts.

First, we consider cohort and outcome definitions. In defining the cohort, the user specifies a SQL script which returns the set $\chi(\mathcal{P})$ expressed as a database table, with rows corresponding to patients in the set. The two primary columns in turn correspond to a universal key identifying patients p, and the outcomes $\Omega(p)$ for each patient. Finally, we augment this table with a third column filled in with the user-defined prediction time T_A. The expected columns returned by a cohort script are described in Table 3.2.

Next, the feature generation scripts $c_1, c_2, \cdots c_n$ are likewise defined by SQL scripts. In these scripts, the user is expected to return tables with each row corresponding to a single tuple (p, c, t) indicating that patient p was assigned code c at time t, with t. Note that the term *code* refers to a generalization of OMOP codes – a user may define any custom function of OMOP codes as a new code in Prediction Library, so

Column	Meaning
`example_id`	A unique integer for each data point in the cohort that can be used for downstream indexing.
`person_id`	An integer identifying each unique patient in the cohort. This integer should be the same `person_id` used in the OMOP CDM's `person` table to allow for cross-referencing and joining additional data
`y`	Binary outcomes per example, represented as integers in $\{0, 1\}$.
`end_date`	The date on which patient data collection is to cease and a prediction is to be made. This is stored and used to eliminate any potential leakage of censored information in downstream processes.

Table 3.2: Required Columns for Cohort Definition

Column	Meaning
`person_id`	An integer identifying the unique patient to whom the feature is relevant to. This integer should be the same `person_id` used in the OMOP CDM's `person` table to allow for cross-referencing and joining purposes.
`date`	A date, represented by entries of SQL data type `Date`, representing when this feature is applicable to a patient.
`feature_name`	An arbitrary string that names the feature represented in each row, ideally utilizing a simple and comprehensible nomenclature.

Table 3.3: Required Columns for Feature Definition

long as this transformation is expressible in the SQL language. Prediction Library automatically adds a join to the user's queries for each c_i to enforce that only rows (p, c, t) where $p \in \chi(\mathcal{P})$ are collected, ensuring that we do not waste runtime finding data for patients who are not of interest to the task at hand. The layout of feature tables is given in Table 3.3.

We create the sparse-tensor data matrix D using the union of all tables generated by the feature generation scripts. The final output to the user is then the cohort, outcomes, sparse-tensor D of data, and the three invertible mappings h_p, h_c, h_t from the indices of the matrix D to patients, codes, and times respectively.

3.2.1 Establishing Data Notation

Using the sparse-tensor representation of our dataset, we can easily access visits, codes, and visit details. Here, we establish notation for each of these elements of the data. We utilize the same notation to describe our modelling algorithms.

We denote the set of visits made by a patient i by $\mathcal{V}_i$ – these are represented by the columns of D whose first index is $h_p^{-1}(i)$. We represent this patient's jth visit in chronological order, whose occurrence time we denote t_j^i, by V_j^i. This visit is represented by the columns of D whose first index is $h_p^{-1}(i)$ and whose second index is $h_t^{-1}(t_j^i)$. We further denote the set of codes assigned during visit V_j^i with $C_j^i \subseteq \mathcal{C}$, where $\mathcal{C}$ represents all codes present in our data. This set of codes is simply the function h_c applied to the third index of all columns of D whose first index is $h_p^{-1}(i)$ and whose second index is $h_t^{-1}(t_j^i)$.

3.2.2 Likelihood of Hospitalization Task Setup with Prediction Library

In this subsection we develop an end of Likelihood of Hospitalization task with Prediction Library to further clarify its usage. This task is a prototypical problem in clinical prediction, and successful early prediction of the need for in-patient treatment can help inform interventions and preventative care. As this task is indeed a focus of the modelling methods discussed in later chapters, we find it particularly informative to clarify in detail the definitions of its constituent parts.

We describe the exact input a user of Prediction Library would need to provide through annotated SQL scripts. In these scripts, we utilize subquery factoring to improve performance as well as the clarity of the code. Prediction Library supports the formatting of SQL scripts using standard python formatting, and as such terms that appear in braces (such as `{training_end_date}`) can be filled in dynamically just as data is to be collected. This significantly streamlines the process of generating several similar prediction tasks with slight differences, such as varying prediction dates to evaluate performance over time.

3.2.2.1 Defining a Cohort and Outcome

In this scenario, we seek to predict if patients will be admitted for inpatient care between three and nine months after a given prediction date. Our cohort consists

of all patients who have sufficient observed medical history to make a meaningful prediction – we quantify this notion by enforcing that all patients are enrolled in the insurance system for at least 95% of days in a user-specified period prior to the prediction date. We measure outcomes by checking for an OMOP code corresponding to an inpatient hospitalization during the outcome observation window – this window begins a fixed user-defined amount of time `{gap}` after the prediction date, and lasts for a user-defined interval `{outcome_window}`.

The actual cohort generation script is shown in Listing 3.1. We explain the functionality of each block of code:

- Block A allows the user to name the cohort table for future reference, and prepares a `with` statement which will allow us to construct subqueries in subsequent steps.

- Block B defines an outcome at a per-patient level. To do so, we use the OMOP CDM's Visit Occurrence table, whose rows comprise all recorded visits made by any patient – this table's `visit_concept_id` column codifies what type of visit a row refers to, with the code 9201 corresponding to inpatient care. We can thus select all rows corresponding to inpatient visits by constraining `visit_concept_id` to be 9201. We also constrain outcome measurements to lie within the desired time window, which starts at time

  `{observation_threshold_date} + {gap}`

  and ends at time

  `{observation_threshold_date} + {gap} + {outcome_window}`.

  The result of this subquery is simply a list of patients for whom we would have a positive outcome, and we later assume all other patients have a negative outcome. Interestingly, we do not explicitly perform the outcome calculation only on patients who meet our filtering criteria – this does not affect efficiency in practice due to subquery factoring, which will optimize the entire query as a

whole to eliminate redundant or unnecessary operations. Users can thus code relatively simple functions to demarcate outcomes without any performance-related disadvantages.

- Block C queries the Observation Period table of the OMOP CDM, which in our dataset records when a patient's healthcare history was being tracked as part of their insurance plan. We retrieve one row for each contiguous interval for which a patient's history was tracked, and truncate these intervals to lie between the prediction date set by the variable `{prediction_date}` and `{observation_threshold_date}`. We next want to select patients whose medical history is available for at least 95% of the days between a desired start date `{observation_threshold_date}` and the prediction date. We check this condition with a simple aggregation of intervals of observation periods by patients

- In Block D, we bring all subqueries together by selecting patients who satisfy the cohort criteria, and augmenting their patient IDs with the outcomes calculated in Block B. Note that patients for whom an outcome is not defined are given a negative label by default. We further add in an index column denoted `example_id` for each patient specific to this task.

```
-- Block A
create table {schema_name}.{cohort_table_name} as

with
    -- Block B
    outcomes as (
        select
            a.person_id,
            1 as outcome
        from cdm.visit_occurrence a
        where
            visit_concept_id = 9201 -- 9201 is the OMOP code for an
   Inpatient Visit
            and
```

```
            a.visit_start_date between
                    date '{prediction_date}'
                     + interval '{gap}'
                    and
                    date '{prediction_date}'
                     + interval '{gap}'
                     + interval '{outcome_window}'
        group by
            a.person_id

    ),
    -- Block C
    count_of_enrolled_days as (
        select
            person_id,
            observation_period_start_date as start,
            observation_period_end_date as finish,
            greatest(
                least (
                    observation_period_end_date,
                    date '{prediction_date}'
                ) - greatest(
                    observation_period_start_date,
                    date '{observation_threshold_date}'
                ), 0
            ) as num_days
        from cdm.observation_period
    ),
    eligible_people as (
        select
            person_id
        from
            count_of_enrolled_days
        group by
            person_id
        having
```

```
            sum(num_days) >= 0.95 * (date '{prediction_date}' - date
    '{observation_threshold_date}')
    ),
    -- Block D
    select
        row_number() over (order by e.person_id) - 1 as example_id,
        e.person_id,
        date '{prediction_date}' as end_date,
        coalesce(o.outcome, 0)::int as y
    from
        eligible_people e
        left join outcomes o
            on o.person_id = e.person_id
    ;
```

Listing 3.1: Cohort and Outcome Generation Script for a Likelihood of Hospitalization Task

3.2.2.2 Collecting Features

In this example, for the purpose of simplicity we seek to collect a dataset consisting only of the recorded conditions associated with each patient, along with the days on which these conditions occurred. Note that the logic used to collect such data can easily be modified to collect other OMOP concept types as well.

As described in Table 3.3, the feature generation script must minimally provide a set of rows with columns corresponding to a patient, a feature name, and the date on which the feature was assigned to the patient. As we make explicit in Listing 3.2, we collect this data by simply querying the OMOP CDM's `condition_occurrence` table, but more complex transformations of the data in this table can also be represented as features. We name features using the OMOP CDM `concept` table, which maps numerical concept identification numbers to English names.

Note that we inner-join the data extracted from the `condition_occurrence` table with the patients who are actually present in the cohort, as represented by the variable `{cohort_table}`. This can easily be inserted automatically using the same name used

Data Structure	Filtering Runtime	Aggregation Runtime	Memory Footprint
List of Data	$O(\|\mathcal{R}\|)$	$O(T_g A\|\mathcal{R}\|)$	$O(\|\mathcal{R}\|)$
3-d Dense Tensor	$O(A)$	$O(T_g A)$	$O(A^3)$
3-d Sparse Tensor	$O(\|\mathcal{R}\|)$	$O(T_g\|\mathcal{R}\| + \|\mathcal{R}\| \log \|\mathcal{R}\|)$	$O(\|\mathcal{R}\|)$

Table 3.4: Memory and Runtime Tradeoffs when Storing Longitudinal Health Data

to generate the cohort for the task at hand, and up to this minor insertion of a table name, the feature generation script is agnostic to the task and the cohort. Thus, feature scripts that collect interesting and useful features can easily be reused and shared between tasks.

```
select
    a.person_id,
    a.condition_concept_id || ' - condition - ' || coalesce (
        c.concept_name, 'no match'
    ) as concept_name,
    a.condition_start_datetime as feature_start_date
from
    cdm.condition_occurrence a
inner join
    {cohort_table} b
on
    a.person_id = b.person_id
left join
    cdm.concept c
on
    c.concept_id = a.condition_concept_id
```

Listing 3.2: Feature Generation Script for Condition History

Chapter 4

Linear Models for Clinical Prediction

Linear models are often highly performant in clinical prediction tasks, especially when input features are carefully chosen to reflect the nuances of medical data. These models have the advantages of simplicity and interpretability, as well as the benefit of being able to be trained and retrained without too high of a computational burden. In this chapter, we discuss ways in which linear models for clinical prediction are constructed, and outline the design of our own model in this class. Our linear model, which is currently in production at IBC, already offers excellent performance on baseline tasks and serves as a launching point for further advances.

4.1 A Basic Linear Model

In prior work, a useful linear baseline has been an encoding of a patient's entire medical history without temporal information [13]. To define this encoding, we recall the data definitions established in Section 3.2.1 – we construct the set of all codes ever assigned to patient i by first encoding the codes assigned during each visit as a multi-hot vector one-hot$(C_j^i) \in \{0,1\}^{|\mathcal{C}|}$ with entries corresponding to each unique code in the dataset. The element of this vector corresponding to the code $c \in \mathcal{C}$ is defined to be

$$\text{multi-hot}(C_j^i)_c = \begin{cases} 1 & c \in C_j^i \\ 0 & \text{otherwise} \end{cases}. \tag{4.1}$$

We combine these multi-hot encodings of each visit with a simple summation, arriving at a final multi-hot encoding multi-hot$(C^i_{\text{all}}) \in \mathbb{N}^{|\mathcal{C}|}$ of the entire patient history defined by

$$\text{multi-hot}(C^i_{\text{all}}) = \sum_j \text{multi-hot}(C^i_j). \tag{4.2}$$

This vector represents counts of how many times each code was ever assigned to patient i.

The basic linear model consists of training a L_2-regularized logistic regression model to predict patient outcomes using multi-hot(C^i_{all}) as input. That is, for each task of interest we learn a vector $w \in \mathbb{R}^{|\mathcal{C}|}$ that minimizes the loss function defined by

$$\mathcal{L}_{\text{Basic Linear}} = \sum_i \ell_{\text{Basic Linear}}(i) + \lambda ||w||_2 \tag{4.3}$$

where the sum is taken over all patients i and the per-patient loss $\ell_{\text{Basic Linear}}(i)$ is defined by

$$\ell_{\text{Basic Linear}}(i) = y_i \log\left(\sigma(w \cdot \text{multi-hot}(C^i_{\text{all}})\right) + (1 - y_i)\left(1 - \sigma(w \cdot \text{multi-hot}(C^i_{\text{all}})\right). \tag{4.4}$$

In these equations, $y_i \in \{0, 1\}$ is the outcome of patient i, λ represents the regularization strength which is yet to be determined, and σ the sigmoid function $\sigma(x) = e^x/(e^x + 1)$.

4.2 Linear Models with Temporal Data

A clear flaw of the basic linear models is that they fail to account for the inherently temporal nature of a patient's medical history. A more nuanced approach would be able to identify narratives over time. This would allow it, for example, to differentiate between the worsening and the improvement of a specific disease by seeing how diagnosis and treatment evolves over multiple visits. In addition, the spacing and density of visits is also of medical importance – a patient with a few visits spaced throughout time is probably more healthy than one who has the same number of

visits tightly packed into a small interval in the recent past. Finally, incorporating temporal information allows us to decay the relevance of very old medical events and up-weight more recent ones, since these will often be more relevant to a patient's future condition.

To incorporate this kind of information into a linear model, we must build features that have temporal dependencies. A natural way to do so that has been successful in past work is to create *windowed* features [43]. In this setting, we construct features by aggregating codes over different temporal windows, which are defined as intervals of dates. This method has been used successfully in the past to predict long-term outcomes such as the onset of Type II Diabetes [43] on data very similar to that used in our work. As both the input data and predictive tasks under consideration are similar, we can confidently ground our linear models in this prior work.

Consider the development of such features for patient i. Given a temporal interval $W = [t_s, t_e]$, where t_s represents a start date and t_e an end date for the window, we find the set of all codes assigned to patient i during the interval W. To do so we first extract the subset of visits

$$\mathcal{V}_i(W) = \{V_j^i \in \mathcal{V}_i | t_j^i \in W\}. \tag{4.5}$$

Note that this can be achieved using a filtering operation on our sparse-tensor representation of longitudinal health data. We subsequently find the set of codes associated with any of the visits in the window:

$$\mathcal{C}_i(W) = \bigcup_{V_j^i \in \mathcal{V}_i(W)} C_j^i. \tag{4.6}$$

We represent the codes present in the window with the multi-hot vector $\text{multi-hot}(C_i(W)) \in \{0,1\}^{|\mathcal{C}|}$ to represent these sets, with the element corresponding to concept $c \in \mathcal{C}$ set equal to 1 if $c \in \mathcal{C}_i(W)$ and 0 otherwise.

To capture the longitudinal nature of claims data, we use multiple windows simultaneously as features. In practice, this means that we concatenate the one-hot

representations corresponding to each window.

We regularize and tune this model to achieve good predictive performance. First, we establish a list $\mathcal{W}_C$ of candidate windows, each of which has an end time equal to the prediction date and start times ranging from 15 to ∞ days before the prediction date – in practice, we use the candidate windows displayed in Table 7.2.2. We select the $n_W = 5$ best windows from all $\binom{|\mathcal{W}_C|}{n_W}$ unique window choices by comparing validation performance across a subset of the data, and call this optimum window set $\mathcal{W}$. In addition, given that this model will have numerous redundant features, we apply heavy L_1 regularization and tune the strength of this regularization.

In all, we train a logistic regression model on the concatenation of n_W temporal windows, which amounts to learning a vector $w \in \mathbb{R}^{n_W \cdot |\mathcal{C}|}$ that minimizes the objective function given by

$$\mathcal{L}_{\text{Windowed Linear}} = \sum_i \ell_{\text{Windowed Linear}}(i) + \lambda ||w||_1 \tag{4.7}$$

where the sum is taken over all patients i and the per-patient loss $\ell_{\text{Windowed Linear}}(i)$ is defined by

$$\begin{aligned} \ell_{\text{Windowed Linear}}(i) &= y_i \log\left(\sigma\left(w \cdot \Big\|_{W \in \mathcal{W}} \text{multi-hot}(C_i(W))\right)\right) && (4.8) \\ &\quad + (1 - y_i)\left(1 - \sigma\left(w \cdot \Big\|_{W \in \mathcal{W}} \text{multi-hot}(C_i(W))\right)\right). && (4.9) \end{aligned}$$

where we denote concatenation of vectors with the symbol $\Big\|$, and as before $y_i \in \{0, 1\}$ is the outcome of patient i, λ represents the regularization strength which is yet to be determined, and σ the sigmoid function $\sigma(x) = e^x/(e^x + 1)$.

Chapter 5

Deep Models for Clinical Prediction

In this chapter, we introduce the principal deep-learning paradigms used in patient-level clinical prediction tasks from prior medical information. These approaches take varying approaches to converting raw medical data from a patient to predictions – at their core however, all of these methods account for the uniquely defining properties of longitudinal health data. We then discuss our novel advances in applying deep learning to clinical prediction through the introduction of a novel architecture and training method which when combined can achieve state of the art performance on several medical prediction tasks with applications to improving care.

5.1 Prior Research

Deep learning techniques offer a path to improving predictive performance for clinical prediction tasks by learning representations of longitudinal health records that capture a patient's medical status and potential future risks. State-of-the-art models in the literature have largely focused on shorter-term prediction over horizons of days or weeks, most notably during a single hospital visit [4, 49, 37, 44, 12], or in the immediate aftermath of a visit [37, 44]. Approaches to longer-term prediction often rely on manually feature-engineering longitudinal health data into patient state vectors [43, 4, 5, 35], as opposed to training end-to-end from raw longitudinal EHR data. Due to this heuristic approach, these methods cannot fully exploit the temporal nature

of EHR data, nor the relationships between clinical concepts. We further find that linear models with well-tuned features are quite competitive with existing end-to-end deep models for long-term prediction, indicating that deep learning successes in other domains have not yet been effectively translated to clinical prediction

Many recent works analyze how deep learning can be applied to clinical prediction [11, 41, 10, 48, 13, 21, 20, 31, 52]. Several approaches use recurrent neural networks (RNNs) to ingest medical records, and achieve excellent performance on tasks like predicting in-patient mortality upon hospital admission [11]. Further refinements add learned imputation to account for missingness [10], and improvements in featurizing time by using architectures like bi-directional RNNs [30] and two-level attention mechanisms to find the influence of past visits on a prediction [13, 26]. Research has also focused on using convolutional neural networks (CNNs) to develop better embeddings of clinical concepts passed into a recurrent model [31], and graphically representing the patient-clinician relationship to augment health record data [52]. Self-attention has also been used in a non-longitudinal manner, to develop relationships between medical features that have already been collapsed over the temporal dimension using recurrent methods [32].

When making predictions with horizons of months or years, the state-of-the-art is still simple, often linear models with carefully chosen features [43, 4]. Recent work exploring deep-learning based approaches to long-term clinical prediction train neural networks directly on features constructed using hand-picked time windows and summary statistics [5] or use denoising autoencoders to pre-process this type of data [35], and do not necessarily beat linear baselines [41, Supplemental Table 1]. Critically, many of these models rely on manual feature-engineering to create representations of the time-series data that forms a patient's medical record rather than learning this structure in tandem with the task at hand.

EHRs represent collections of longitudinal patient-level medical encounter data. As discussed in 2 for each patient in an EHR system we receive a time series of *visits* – single continuous interactions of a patient with the healthcare system – and *codes* – the medical events occurring during each visit. As such, machine learning

models operating on claims data must learn from their rich temporal and conceptual structure to create a patient-level representations. The following key properties of claims data inform the desiderata for model architecture in this domain:

- **Sparsity of Features**: Claims data is extremely sparse. The set of codes is large, and the overwhelming majority of codes do not apply to a given patient at a given time.

  As an example, consider the naive expansion of our insurance claims dataset into a large three-dimensional tensor whose axes correspond to time (measured in days), codes, and patients respectively. If we set the element corresponding to day d, code c and patient p to 1 if patient p had code c assigned to them on day d, and set this element to zero otherwise, only 1 in 10^6 elements of the tensor would be nonzero. Thus, it is completely infeasible from the perspective of memory and computational constraints to operate on dense representations of EHR data, and it is critical that architectures be able to efficiently process sparse input data. In prior work, this has been achieved by manipulating sparse binary multi-hot vectors representing active codes [11, 13], as these vectors can effectively be represented as lists of indices of nonzero vector elements.

- **Code-level Permutation Invariance**: The unique symmetries and asymmetries of claims data must be preserved by an effective architecture. The set of codes assigned during a visit is not ordered and thus any operation on groups of codes must be invariant to permutations of its inputs. While brute-force techniques like randomization of inputs can achieve this invariance in expectation, a nuanced approach is needed to efficiently implement a transformation layer that is intrinsically permutation-invariant.

  We note that the characterization of codes as permutation-invariant fails to account for the common clinical paradigm of a single code representing the main theme of a visit, and other codes acting as supporting evidence and background for this primary theme. However, this level of granularity in clinician decision making is not captured in our EHR data and as such cannot be directly encoded.

As such, allowing an architecture to learn how much importance to put on each code and how to contextualize codes for a specific task is necessary.

- **Visit-level Temporality**: Visits are ordered and augmented with timestamps corresponding to when care was provided. Models must be able to process a highly irregularly-spaced time series of events, since care is often administered in short bursts punctuated by long gaps. Indeed, the time between visits made by a single patient can vary from years to days, based on their medical needs, and both short-term and long-term dynamics can be important. Visits that are close temporally can effectively represent the same overall medical event being treated over multiple sessions, while two temporally distant visits may be deeply connected as manifestations of a chronic disease or untreated underlying condition. This has been previously approached by discretizing time to allow for uniform time steps [42], bucketizing time [43] to create categorical representations of when events occurred in a timeline, and treating time as a continuous covariate in models [26, 12].

5.2 Representing Longitudinal Health Data with Self-Attention

We propose a novel architecture that addresses the challenges described in Section **??** by building upon self-attention architectures [49], which have seen success in natural language processing [18] and time-series analysis [50, 27]. Self-attention mechanisms allow each element of a sequence to extract context from, or *attend* to, other sequence elements. What makes this paradigm particularly appealing is that any element can directly attend to any other element regardless of their separation within the sequence, which means that long-term dependencies are captured well.

This is in contrast to recurrent or convolutional methods, where information must flow through many layers of computation to get from one sequence element to a distant other. In asymptotic terms, in order to move contextual information from a

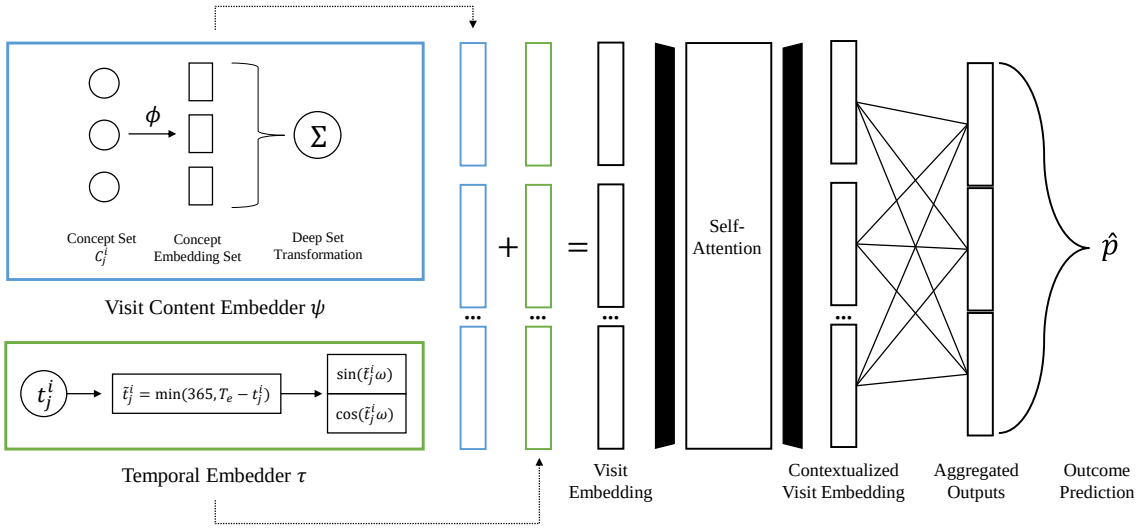

Figure 5-1: SARD Architecture for Longitudinal Claims Data

patient's i^{th} visit to the same patient's j^{th} visit, a recurrent or convolutional model would need to move this information through $O(|i-j|)$ edges of the neural network's computational graph. On the other hand, a self-attention based architecture would only need to move the same information through $O(1)$ edges of the neural network's computational graph. Our model, which we denote as Self Attention with Reverse Distillation, or SARD, is able to cleanly and interpretably associate visits over a variety of timescales.

We use a set encoding approach to address the challenge of sparsity and the need to represent a set of data observed at each visit, and a self-attention based architecture to allow any visit's embedding to interact with another visit embedding, ensuring that we can capture temporal information and dependencies. An overview of the architecture is provided in Figure 5-1.

We denote the set of visits made by a patient i by $\mathcal{V}_i$, and represent this patient's jth visit by V_j^i. We further denote the time of visit V_j^i by t_j^i and the set of codes assigned during visit V_j^i with $C_j^i \subseteq \mathcal{C}$, where $\mathcal{C}$ represents all codes present in our data. We use this general framework for our architecture so that it is extensible to a variety of clinical settings and tasks, where specific types of information may not be readily available.

The architecture of SARD consits of the following components:

- **Code Embedding**: We adapt the method of Choi et al. [14] to generate an initial concept embedding map $\phi : \mathcal{C} \to \mathbb{R}^{d_e}$ which takes each code in our data and maps it to a representative vector of real numbers. In order to generate these embeddings, we use a skip-gram architecture [34]. In this embedding paradigm, we present a model with a single concept c and train it to predict other concepts that contextualize c – in the original use case of natural language processing and word embeddings, the context of a word was chosen to be other words occurring in the same sentence, and a natural extension of this idea to the clinical domain is to make the context of concept c the set of other codes occurring during the same visit. However, prior research [14] has established that this is not optimal, since information relevant to a code can occur in other visits that are temporally nearby. As such, we define the context of code c to be any code that occurs within 90 days of c, and train a skip-gram model using the Gensim Library [45]. Note that in practice, this embedding is learned only using data in a training window to prevent label leakage.

- **Visit Content Embedding** The vector representation $\psi(V_j^i) \in \mathbb{R}^{d_e}$ of each visit is calculated in a manner inspired by the Deep Set paradigm, which develops a general approach to the problem of embedding a set of objects drawn from a large space in an efficient manner [51]. Critically, sets are unordered collections of objects and as such embeddings of a set must be invariant to permutations of set elements. As established in [51], any embedding satisfying this property can be written as a nonlinear function f applied to the sum of another nonlinear function g applied to each set element s of some set S. That is, the embedding of S can always be expressed as $f(\sum_{s \in S} g(s))$.

  In our case, we already apply a nonlinear transformation ϕ to each code in order to embed it as a real-valued vector, and we can utilize the downstream components of the SARD architecture to act as the nonlinearities encapsulated

in the function f. As such, we define

$$\psi(V_j^i) = \sum_{c \in C_j^i} \phi(c), \tag{5.1}$$

and we note that this formulation has the desired property that the embeddings ψ are invariant to re-orderings of the codes in C_j^i.

- **Temporal Embedding**: We note that a self-attention mechanism does not explicitly encode an order of events, and instead relies on the use of a *temporal embedding*, which represents when an event occurs similarly to how a content embedding encodes an event's meaning. In the case of SARD, noting that visits do not occur in regular intervals, our choice of time embeddings must be able to capture different timescales effectively. We embed the time of each visit as a real-valued vector in $\mathbb{R}^{d_e}$ using a sinusoidal embedding, which has been found in prior research to be an effective way to encode temporal information in self-attention based architectures [49].

  Using this framework, we create temporal embeddings

$$\tau(V_j^i) = \sin(\tilde{t}_j^i \omega) || \cos(\tilde{t}_j^i \omega), \tag{5.2}$$

  where $\tilde{t}_j^i = \min(365, T_A - t_j^i)$ and T_A represents the prediction date. This allows us to measure time relative to the prediction date. Note that we denote the concatenation of two vectors with $||$, ω denotes a length $d_e/2$ vector of frequencies in geometric progression from 10^{-5} to 1, and the functions sin and cos are applied element-wise to their vector arguments.

  The definition of the frequency vector ω as a geometric progression of frequencies means that various timescales are inherently incorporated into the temporal embedding. Indeed, given a sufficiently large number of frequencies (or equivalently a large enough embedding dimension d_e), arbitrary functions of time can effectively be learned and used by the model through the combination of

different frequency-based functions into Fourier series.

- **Self-Attention**: Following the predominant practice established by prior research into self-attention architectures [49], we add $\psi(V_j^i)$ and $\tau(V_j^i)$ to create final real-vector-valued encodings that represent both the content and timing of visits. The next step is to develop architectural elements that allow these visit representations to contextualize each other in a patient's overall history. To do so, we utilize multi-headed self-attention [49] with $L = 2$ self-attention blocks and $H = 2$ heads.

  In each layer of each head of the self-attention mechanism, every visit *attends* to every other visit. To do so, we first perform three affine transformations on the input embeddings, which for the first layer are $\psi(V_j^i) + \tau(V_j^i)$ for each visit V_j^i. These transformations produce vectors k_j^i, q_j^i and v_j^i, which represent a *query*, *key* and *value* associated with each visit. Loosely inspired by information retrieval algorithms, we find the contextualized embedding of visit V_j^i by computing raw attention weights

$$w_{j\ell}^i = \frac{q_j^i \cdot k_\ell^i}{\sqrt{d_e}}, \tag{5.3}$$

  which represents in a single real value how much context from visit V_ℓ^i should be incorporated into visit V_j^i based on their query and key vectors respectively. We next normalize these weights via softmax to get

$$\tilde{w}_{j\ell}^i = \frac{e^{w_{j\ell}^i}}{\sum_{\ell'=1}^{n_v} e^{w_{j\ell'}^i}}. \tag{5.4}$$

  We next incorporate the value vectors for each visit. These vectors are trained to represent the actual contextual information contained within a visit, and we determine an overall contextualized representation for visit V_j^i by taking the weighted sum

$$\sum_{\ell=1}^{n_v} \tilde{w}_{j\ell}^i v_\ell^i. \tag{5.5}$$

This process of determining how much context should be given to each visit is then repeated at each layer using the contextualized embeddings generated by the previous layer as inputs, recalling that in the first layer, we simply use the summed outputs of the temporal and content embedders $\psi(V_j^i) + \tau(V_j^i)$ as the embedding for each visit V_j^i Each head, in turn, performs the algorithm described above in parallel, allowing for multiple different types of context to be collected simultaneously.

Residual connections are used between layers of the self-attention mechanism to improve performance. The outputs of each head are concatenated to create final, contextualized visit representations $\tilde{\psi}(V_j^i)$. The For efficiency, we truncate to the $n_v = 512$ most recent visits, and add padding for patients with less than n_v visits – pad visits are represented by a learned vector $\phi_{PAD} \in \mathbb{R}^{d_e}$. We apply dropout with probability $\rho_d^t = 0.3$ after each self-attention block to prevent overfitting. This approach allows any visit to attend to any other, so longer-range dependencies of clinical interest can be learned.

- **Prediction Head**: The prediction head is the final stage of the architecture, and it returns an estimated probability of the target event using the outputs of the self-attention mechanism. The final visit representations $\tilde{\psi}(V_j^i)$ each represent contextualized views of patient health at a certain point in time. In previous applications of self-attention to classification in the natural language domain [18], an extra dummy token is added the the end of a sequence to be classified, and this token learns to attend to and aggregate whatever data is necessary for prediction.

  We instead opt for a new approach in which we use a learned output of all contextualized visits instead of a single specialized summary visit. This allows us to have a shallower transformer architecture, and to have more redundancy in the model. The final visit representations $\tilde{\psi}(V_j^i)$ are combined through an aggregation of $n_m = 10$ separate learned linear combinations of visits.

  For any patient i, this calculation proceeds as follows: we denote the kth of

n_m weights for the visit V_j^i by the learnable real number $a_j^k \in \mathbb{R}$. Then, we calculate a final representation of patient state by finding

$$\Big\|_{k=1}^{n_m} \sum_{j=1}^{n_v} a_j^k \tilde{\psi}(V_j^i), \tag{5.6}$$

where $\|$ denotes the concatenation of vectors. Then, dropout is applied with probability $\rho_d^h = 0.05$ and a ReLU non-linearity is applied element-wise. Finally, a densely connected layer maps the resulting vector to a single real value, to which the sigmoid function is applied to obtain an estimated probability $\hat{p}(i)$ that patient i's outcome is positive.

5.2.1 Interpreting the Predictions of a SARD Model

As the predictions made by our deep model may be used for clinical decision making, it is critical that the model's decisions can be interpreted. There are two primary components that must be understood to make sense of a prediction made by SARD – the self-attention mechanism and the prediction head.

In the self-attention mechanism, we can explicitly trace the flow of information from one visit to another, thereby identifying how different clinical encounters contextualize each other. Understanding the action of the prediction head requires the use of state-of-the-art interpretability techniques, but is tractable since the prediction head is simply a feed-forward neural network.

In interpreting the prediction head, we seek to determine which contextualized visits output by the self-attention layers of SARD are most influential to the final prediction. To do so we use the DeepExplainer function from the SHAP package [29]. The inputs to the prediction head are contextualized visits, represented as a d_e-dimensional vectors; for a given patient, SHAP assigns a positive or negative score to each component of each visit vector, indicating whether that component added to or subtracted from the predicted probability of death. We represent a visit's overall importance to the prediction by computing the sum over all vector components of the absolute value of SHAP scores for each visit.

We can then observe which un-contextualized visits from the input to the self-attention layers were strongly attended to by these influential contextualized visits to determine how SARD merges information across timescales. This gives us a qualitative yet thorough method to find the key drivers of SARD's predictions. We demonstrate this technique in practice in Section 7.3.4, as a confirmation of our experimental results.

Chapter 6

Learning with Reverse Distillation

In this chapter, we discuss *reverse distillation*, a novel model initialization technique. We use reverse distillation to initialize a deep model for binary classification, or equivalently binary prediction, to mimic a simpler proxy. Reverse distillation offers several potential advantages in terms of model performance. First, by learning a robust way to replicate a proxy model, a deep model can learn robust and redundant ways to represent features of the proxy model, thereby improving generalization. Next, assuming that we can successfully train a complex deep model to generalize in the same way as the proxy model, we can effectively initialize the deep model to a point in the parameter space which is potentially closer to the optimum, or in a region of the landscape of the objective function where optimization is easier – this is similar to pre-training approaches that lead to improved model performance [18]. In addition, if the proxy model is well-regularized, it effectively performs feature-selection, and the deep model may be able to learn this feature selection in a soft manner, allowing it to generalize better as well. Finally, reverse distillation can act as a form of regularization, by keeping the weights of the deep model close to the set of points in the parameter space which make the deep model generalize similarly to the proxy model.

6.1 Reverse Distillation Training Procedure

We formally define the reverse distillation training algorithm. We seek to train a binary prediction model $f_\theta : \mathcal{X} \to [0, 1]$ parametrized by θ which maps from a domain $\mathcal{X}$ of data to a probability value. We also are given a linear proxy model $g_w : \mathcal{X} \to [0, 1]$ defined by $g_w(x) = \sigma(w^T \xi(x))$, where σ is the sigmoid function and ξ is a fixed *feature engineering* transformation $\xi : \mathcal{X} \to \mathbb{R}^d$ based on heuristic domain knowledge. In the clinical domain, for example, ξ may represent the calculation of various risk scores from raw patient data $x \in \mathcal{X}$, or the process of generating windowed features as in our windowed linear models in Section 4.2.

Even though f_θ may be a large, highly-parametrized model, g_w may perform better on prediction tasks when compared to f_θ trained in the traditional manner of optimizing θ to minimize a cross-entropy loss between predicted and true outcomes on a training set. This can be the result of the proxy model's ability to select features and avoid overfitting through regularization of w, and the quality of the transformation ξ which may represent complex insights and domain knowledge. As such, we seek to initialize f_θ to mimic the outputs of g_w in order to benefit from the structure and performance of the linear model while allowing for further data-driven improvements.

We interpret predictions $f_\theta(x)$ (resp $g_w(x)$) as indicating that the distribution of the label for data point x is $\mathbf{B}(f_\theta(x))$ (resp $\mathbf{B}(g_w(x))$), where $\mathbf{B}(p)$ indicates a Bernoulli distribution with success parameter p. We perform reverse distillation by training our deep model by optimizing over θ a loss function defined by

$$\ell_{\mathrm{RD}}(x) = D_{\mathrm{KL}}\left(\mathbf{B}(g_w(x)) || \mathbf{B}(f_\theta(x))\right). \tag{6.1}$$

This algorithm is inspired by the standard knowledge distillation paradigm [22], in which a simpler model is trained to mimic a complex model. A notable high-level difference between these two methods is that while knowledge distillation is often most useful in cases where a model must be made smaller and more efficient for deployability reasons and can therefore be treated as a *post-processing* method, reverse distillation is instead an integral part of the training process for the model f_θ and serves as a

pre-processing method.

To fine-tune f_θ, we make use of both the true label $y(x) \in \{0, 1\}$ and the prediction $g_w(x)$, combining a cross-entropy loss versus the true label and the discrepancy between g_w and f_θ with a hyperparameter α to get a loss function

$$\begin{aligned} \ell_{\text{tune}}(x) &= -\left(y(x) \log f_\theta(x) + (1 - y(x)) \log(1 - f_\theta(x))\right) && (6.2) \\ &\quad -\alpha \left(g_w(x) \log f_\theta(x) + (1 - g_w(x)) \log(1 - f_\theta(x))\right). && (6.3) \end{aligned}$$

Note that the second term in ℓ_{tune} is equal to the KL divergence between $g_w(x)$ and $f_\theta(x)$ up to an additive constant, which we remove to allow α to solely represent the weight placed on differences between $g_w(x)$ and $f_\theta(x)$.

6.2 Synthetic Data Experiments for Reverse Distillation

To help further empirically justify when and how reverse distillation works, we turn to experiments with synthetic data designed to mimic the distinct properties of the kind of data found in domains such as clinical prediction using electronic health records. In particular, we are interested in data where:

- The data is high-dimensional but only a small fraction of these features are useful for any specific downstream task.
- The data is not fully separable, even in the limit of infinite data.

With these two properties in mind, synthetic data for a binary classification problem is generated as follows:

- First, two centers c_0, c_1 are chosen in $\mathbb{R}^d$, with a separation of $||c_0 - c_1|| = \gamma$. We shift the clusters so that the origin is exactly between the two centers.
- Next, for each of N training points:

– We draw a label y for the point from a Bernoulli distribution with parameter ρ.

– We associate K features with the point. The first βK are drawn as iid Gaussian RVs with mean c_y and unit variance. The remainder are uninformative features drawn as iid Gaussians with mean 0 and unit variance.

We can manipulate the fraction of useful features and the separability of the classes by varying β and γ respectively. Our experiments are designed to find when reverse distillation is successful in excess of a simple feature selection procedure – the hypothesis is that reverse distillation would put weight on features similar to those chosen by the underlying linear model, but in a 'soft' and more robust way. As such, the baseline we choose to compare to is a deep model trained only on the features that are not zeroed out by a L_1-regularized linear model.

We note that other, more complex feature selection baselines are possibilities. However, feature-selection in general is straightforward to implement in this synthetic model – one can simply slice out the features chosen by a procedure. With longitudinal medical data, we are not just selecting features temporal contexts as well, and it is not possible to iterate over all such selections. As such, reverse lets us do a 'soft' feature selection over a very complex space of time-series features.

Concretely, we define four procedures whose performance we compare:

- Reverse Distill: We first train an L_1-regularized logistic regression on a synthetic binary classification dataset, tuning the regularization with a validation set to maximize AUC. We collect the predictions $p_{LR}(x)$ made by the linear model at each training point x. Next, a multi-layer perceptron (MLP) with two densely connected layers with ReLU activation, followed by a sum and sigmoid activation to return a probability is initialized randomly and trained until convergence to minimize the KL-divergence between its predictions $\text{MLP}(x)$ and

$p_{LR}(x)$. Finally, this MLP is fine-tuned by minimizing the loss

$$\begin{aligned} \ell_{\text{tune}}(x) &= -\left(y(x)\log f_\theta(x) + (1-y(x))\log(1-f_\theta(x))\right) && (6.4)\\ &\quad -\alpha\left(g_w(x)\log f_\theta(x) + (1-g_w(x))\log(1-f_\theta(x))\right). && (6.5)\end{aligned}$$

where α is a hyperparameter tuned on a validation set.

- Standard Neural Network: We create a multi-layer perceptron, whose action we again denote by MLP$(\cdot)$ using the same architecture as Reverse Distill, and train it until convergence to minimize the loss

$$\begin{aligned} \ell_{\text{NN}}(x,y) &= \text{xent}(\text{MLP}(x), y) && (6.6)\\ &= -\left(y(x)\log \text{MLP}(x) + (1-y(x))\log(1-\text{MLP}(x))\right) && (6.7)\end{aligned}$$

- Feature Selection by L_1 regression: We first train an L_1-regularized logistic regression as in Reverse Distill. Denote the weights of this model by w_i – we define a feature selection function $f_R(x) = \langle x_i \rangle_{\{i|w_i \neq 0\}}$ which takes a feature vector x and creates a new vector whose components correspond to the elements of x which would not be zeroes out by the regularized logistic regression. We create an MLP using the same architecture as Reverse Distill, and train it until convergence to minimize the loss

$$\ell_{L_1 \text{ selection}}(x,y) = \text{xent}(\text{MLP}(f_R(x)), y) \quad (6.8)$$

- Feature Selection by Oracle: We define a feature selection function $f_O(x) = \langle x_i \rangle_{\{i|\texttt{feature i is relevant}\}}$ which takes a feature vector x and creates a new vector whose components correspond to the βK elements of x which are actually relevant for prediction. We create an MLP using the same architecture as Reverse Distill, and train it until convergence to minimize the loss

$$\ell_{\text{oracle selection}}(x,y) = \text{xent}(\text{MLP}(f_O(x)), y). \quad (6.9)$$

This model reflects an optimal feature selection procedure only possible with full knowledge of the generative process for the data, and should beat all other baselines.

We compare the differences in median AUCs on out-of-sample data between Reverse Distill and the other three models to investigate when reverse distillation is useful. Unless explicitly varied, we hold the data generation parameters at $K = 200, \gamma = 0.5, \rho = 0.05, \beta = 0.02$:

- We first investigate how the separability of the data affects the performance gains of reverse distillation by varying γ. We expect that at extremely low separability, no model will be able to do well, and at high separability all models will do equally well. Between these two extremes, we expect reverse distillation to outperform baselines. This is confirmed by our experimental results, as visualized in Figure 6-1

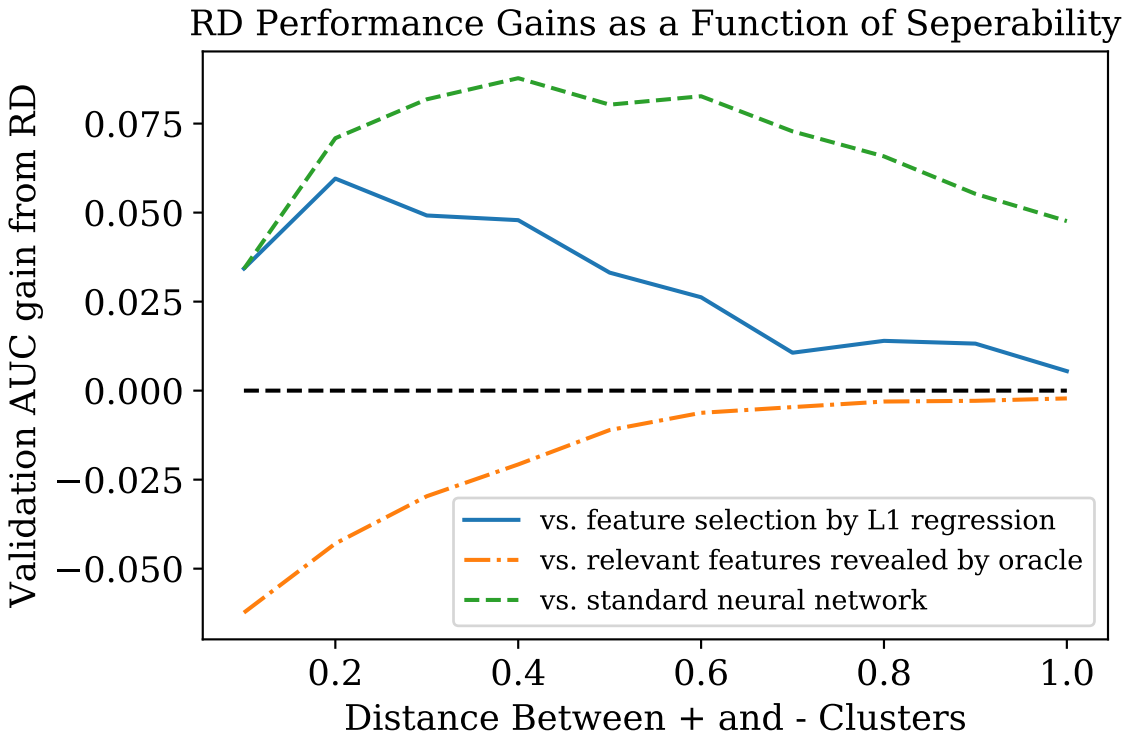

Figure 6-1: Reverse Distillation performance gains as a function of class separability

- We next investigate how the sparsity of useful features affects the performance gains of reverse distillation by varying α. We expect that as α decreases and we see less useful features, that reverse distillation will be more useful since it can make nuanced soft feature selections that can greatly help downstream performance. This is confirmed by our experimental results, as visualized in Figure 6-2

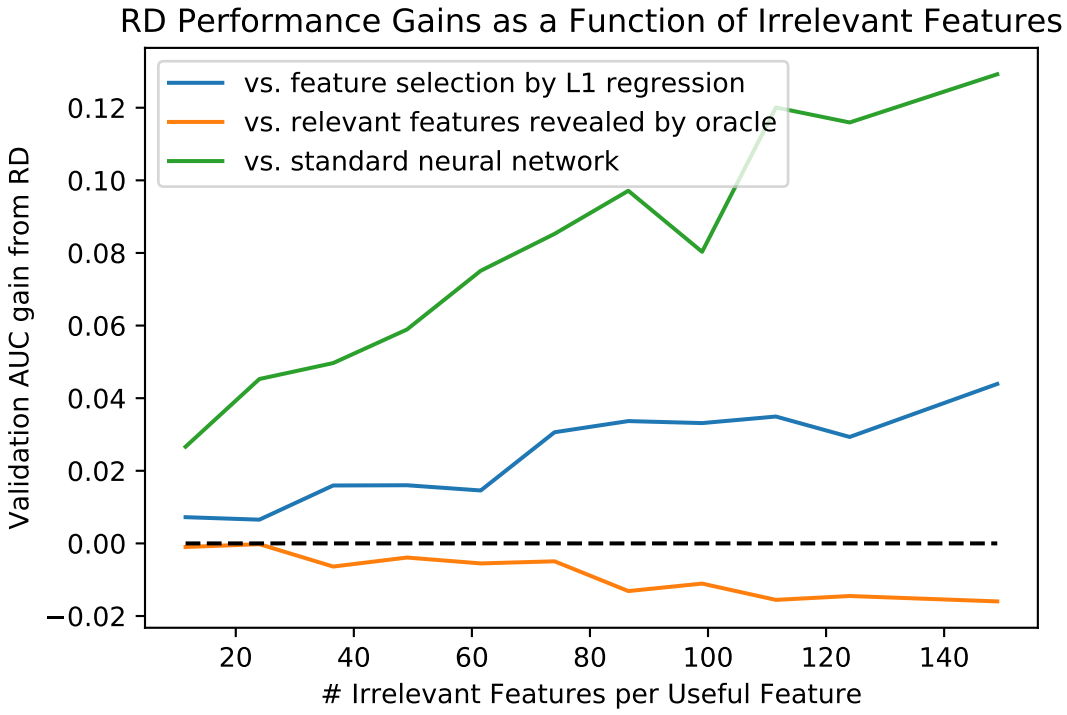

Figure 6-2: Reverse Distillation performance gains as a function of sparsity of useful features

- We finally find that reverse distillation benefits from having more training data, which agrees with the intuition that a deeper, more nonlinear model will have a worse sample complexity. That being said, when an abundance of data is available reverse distillation is able to increase its edge over a baseline feature selection method. This is verified by our experimental results shown in Figure 6-3

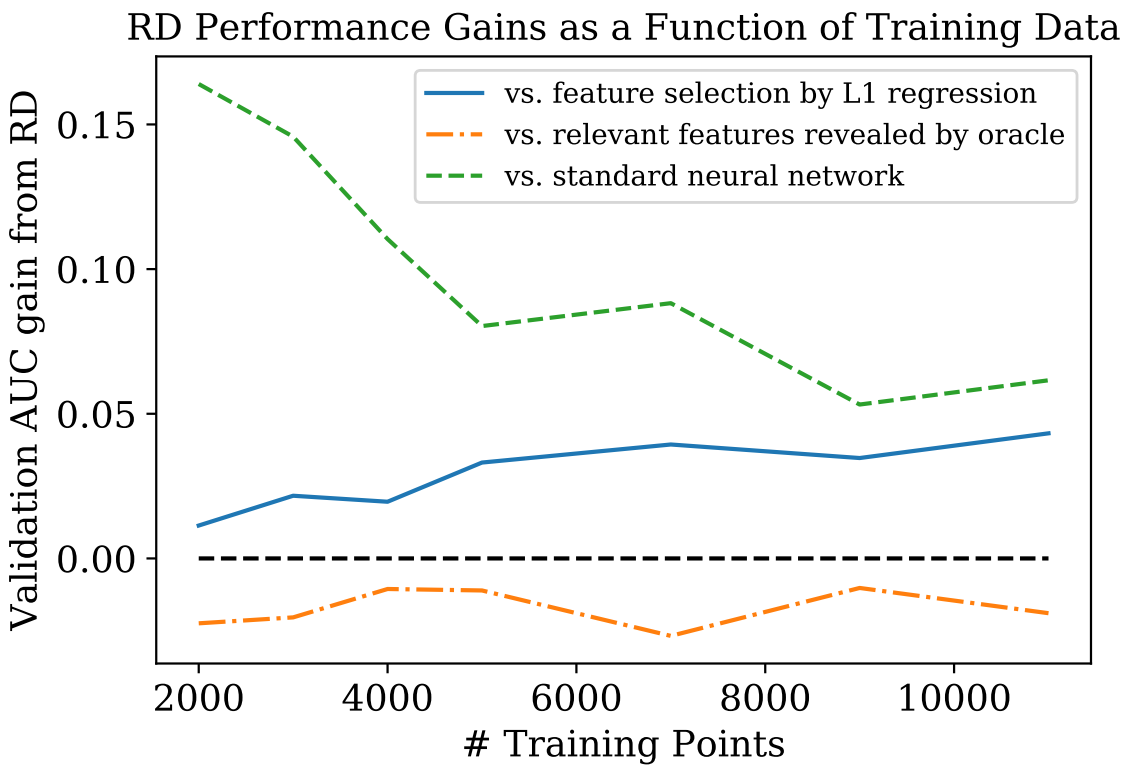

Figure 6-3: Reverse Distillation performance gains as a function of amount of training data

6.3 Training SARD with Reverse Distillation

We seek to utilize reverse distillation as the primary training method for the SARD deep architectures for clinical prediction described in Chapter 5. Since the windowed linear models described in Section 4.2. are highly performant, we use these regressions as proxy models $g_w(x)$. As such, in this case we define $\mathcal{X}$ as the set of all possible patient EHR data, $x \in \mathcal{X}$ as an individual patient's EHR data, and the function ξ as the process of collecting codes assigned to a patient from the data x over different temporal windows and gathering them into a vector representation, as described in detail in Section 4.2.

Ideally, reverse distillation would train a SARD model f_θ to generalize in the same way as the windowed linear proxy model g_w. Adopting the notation of Chapter 5, we find that it is possible to construct a set of weights that allow SARD and a windowed linear model to make identical prediction for all possible inputs:

Lemma 6.3.1. *In the limit $d_e \to \infty$ and for an appropriate choice of ω, SARD can identically replicate a windowed linear model.*

Proof. The crux of the argument is that we can express a filter of the form $[[t_j^i < T]]$ for any T as a linear combination of the elements $\tau(V_j^i) = sin\left(t_j^i\omega\right)||cos\left(t_j^i\omega\right)$, with weights determined as Fourier series coefficients. This allows SARD to replicate the windowed feature vectors of the linear model.

More precisely, we show that a single self-attention head can generate the vector multi-hot$(C_i(W))$ for a given window $W = [T_A - T, T_A]$, thus implying that several self-attention heads' concatenated output can generate the concatenation of several multi-hot$(C_i(W))$ vectors.

Set the embedding function $\phi(c)$ to simply return a one-hot encoding of the code c, concatenated with d_e zeros. We further set g to be an identity function. Then for all i, j, $\psi(V_j^i)$ will be a multi-hot binary vector whose nonzero elements correspond to the codes in C_j^i.

We note that our time embedding per visit will be $\tau\left(V_j^i\right) = sin\left(t_j^i\omega\right)||cos\left(t_j^i\omega\right)$. We set the first $|\mathcal{C}|$ elements of ω to zero, so that visit embedding $\psi(V_j^i) + \tau\left(V_j^i\right)$ will be fully separable component-wise into a multi-hot vector of codes and a time embedding.

The self-attention mechanism will use a linear map from $\psi(V_j^i) + \tau\left(V_j^i\right)$ to three vectors k_j^i, q_j^i, v_j^i called the key, query and value vectors respectively, and create the contextual embedding $\sum_{j'=1}^{n_v}\left(q_j^i \cdot k_{j'}^i\right)v_{j'}^i$ for visit V_j^i. We allow v_j^i to simply be the multi-hot encoding $\psi(V_j^i)$. Note that since $\psi(V_j^i)$ and $\tau(V_j^i)$ have different nonzero components that this can be achieved by a simple matrix multiplication from $\psi(V_j^i) + \tau\left(V_j^i\right)$.

Next, we create appropriate length-1 key and query vectors. We define $k_j^i = q_j^i = [[t_j^i < T]]$, and under this definition the contextual embedding of every visit V_j^i where $t_j^i < T$ will become $\sum_{j'=1}^{n_v}[[t_{j'}^i < T]]v_{j'}^i$, which is a multi-hot vector whose nonzero entities correspond to all codes seen in the window of the past T days.

It remains to show how we would construct $k_j^i = q_j^i = [[t_j^i < T]]$ as a linear transformation of $\psi(V_j^i) + \tau\left(V_j^i\right)$. We do so by invoking a Fourier analysis argument.

Let P be the length of the interval from the first event in the dataset to T_A. Then, $[[t_j^i < T]]$ can simply be represented as a function of period P with value 1 in $[0, T]$ and 0 in $[T, P]$, which in turn can be represented as a Fourier series with coefficient $\frac{2}{n\pi}\sin^2(\frac{n\pi T}{P})$ corresponding to $\sin(\frac{2n\pi t}{P})$ and coefficient $\frac{1}{n\pi}\sin(\frac{n\pi T}{P})$ corresponding to $\cos(\frac{2n\pi t}{P})$, Thus, for appropriately chosen ω that includes values of the form $2n\pi/P$, we can recover an arbitrarily good approximation of $[[t_j^i < T]]$, thus allowing us to use a single self-attention head to mimic a single windowed feature vector as passed into the linear model.

Using multiple self-attention heads, we can obtain the concatenation of several windowed feature vectors, and passing these through the prediction head allows us to fully replicate the functionality of the linear model using the deep model. We additionally note that this lemma holds even with a single self-attention layer.

□

This result increases our confidence in our choice of architecture and its ability to generalize and improve beyond a linear model. For example, windows of the form $[[t_j^i < T]]$ implied by the linear model might be inferior to a more complex filter in the time domain. These complex features, however, can be learned by a SARD model. We further verify that SARD does indeed successfully generalize in the same way as the linear proxy model after reverse distillation in our experimental analyses in Chapter 7.

Chapter 7

Experimental Analyses

In this chapter we discuss the methods by which we evaluated our modelling processes for clinical prediction using longitudinal health data. We first develop and precisely define several experimental tasks which we use for evaluation, then explain in detail how our linear and deep models were trained to produce predictions for each task. Our collected performance data allows us to make nuanced comparisons of different algorithms and to evaluate their relative efficacies.

7.1 Designing Experimental Tasks

We set up example clinical prediction tasks using the Prediction Library framework outlined in Chapter 3. As explained in that chapter, to define a task we require a cohort, and an outcome for each member of the cohort. Our choice of task was inspired by our collaboration with Independence Blue Cross (IBC) – their primary goal was to develop models for longer-term clinical prediction in order to decide which patients could benefit from meaningful interventions to prevent or ameliorate the effects of a potential future issue.

Three tasks were set up following the guiding philosophy of making predictions that could lead to medical intervention:

- The *End of Life (EoL)* prediction task, where we estimate patient mortality over a six-month window starting three months after the prediction date. Such

predictions are critical for providing adequate palliative care.

- The *Surgical Procedure (Surgery)* prediction task, where we predict if a patient will require any surgical procedure over a six-month window starting three months after the prediction date. In this task, we utilize the hierarchy of OMOP concepts as described in 2. By finding all descendants of the high-level procedure concept of `surgery`, we can find all codes which correspond to surgical procedures and thereby mark as positive the patients with any such code assigned to them during the target six-month period. By making predictions of the need for a surgical intervention well in advance of the intervention itself, clinicians can try to intervene in a potentially less invasive and safer manner immediately, leading to better outcomes for patients.

- The *Likelihood of Hospitalization (LoH)* prediction task, where we predict if a patient will require inpatient hospitalization over a six-month window starting three months after the prediction date. We are again able to define outcomes for this task using OMOP concepts, in particular we can simply check if the visit-type concept associated with any visit made by a patient during the target window is the concept `inpatient visit`, and if so label this patient as positive for the Likelihood of Hospitalization task. Successful predictions of hospitalization allow for early interventions that could mitigate the need for potentially invasive, uncomfortable or costly inpatient care.

For all three tasks, we focus on Medicare patients. In addition, in order to ensure that patients in our dataset had sufficient medical records to learn from, we created a cohort of patients whose medical history was sufficiently detailed for us to feel confident in making a data-driven prediction. Our inclusion criterion was that patients are enrolled in a Medicare insurance plan for all of the days in the one-year period leading up to the prediction date, which can be done by utilizing the *payer_plan_period* table of an OMOP CDM.

We receive varying amounts of data per patient, and note that the amount of data a patient has is in itself an interesting indicator of health; for example, a patient with

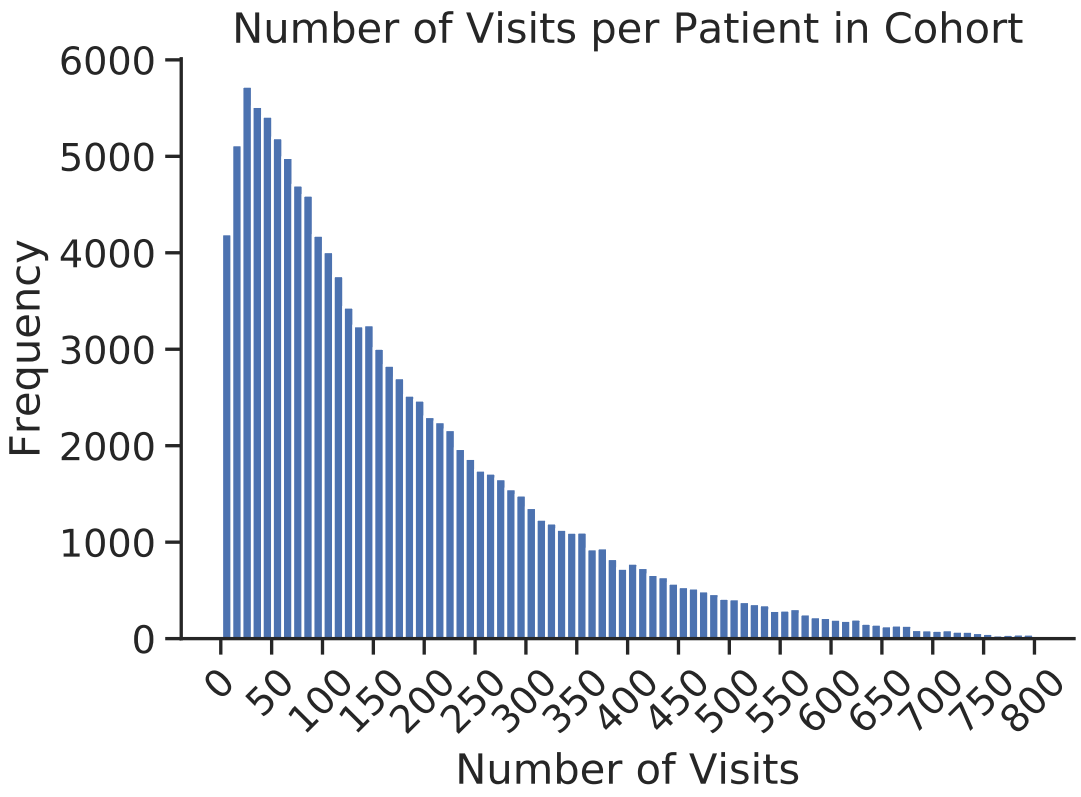

Figure 7-1: Histogram of the number of visits per patient. We clip the histogram at 800 visits, though a small subset of patients (0.4%) have more visits. Histogram buckets have a width of 10 visits.

a long medical history with very few visits may be inferred to be in better health, as they require less medical attention. We next quantify the distributions characterizing the amount of information we have per patient. As shown in Figure 7-1, the length of a patient's history, as measured from the time of their enrollment into an insurance plan tracked by our dataset to prediction time, ranges from 1 to 11 years, with a mean of 7.4 years – the minimum of one year results from the explicit exclusion of patients who entered a tracked insurance plan within one year of the prediction date. Additionally, as shown in Figure 7-2, the number of visits, or unique days during which an interaction with the healthcare system took place, ranges from 1 to $1,616$ per patient with a mean of 175.

As we developed models, we treated End of Life as a prototypical task for model development. Once a model architecture was decided upon up to the selection of

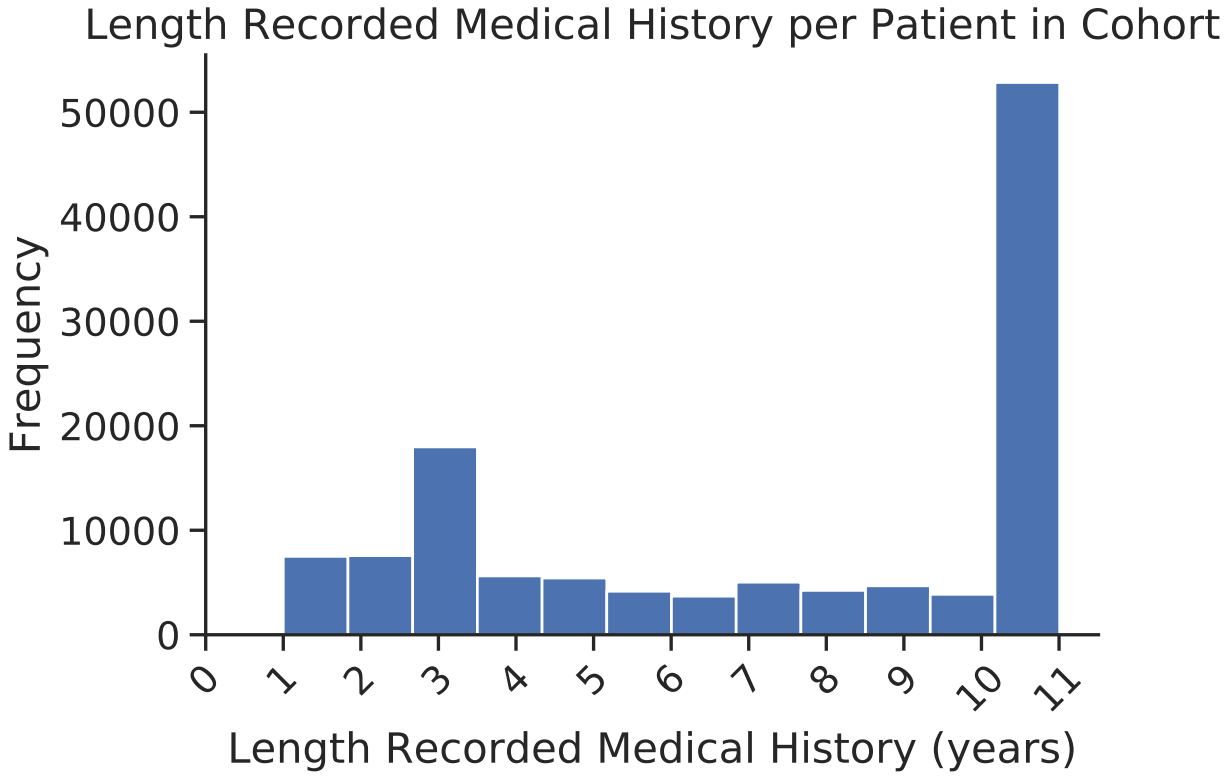

Figure 7-2: Histogram of recorded medical history length per patient.

hyper-parameters and training of weights, we further evaluated the model on the Surgical Procedure and Likelihood of Hospitalization prediction tasks in order to confirm that our techniques generalized to a broader breadth of problems. Ultimately, we seek to be confident in the ability of our model pipelines to be useful for *any* clinical prediction task, which thanks to Prediction Library can easily be set up in terms of an arbitrary cohort and outcome definition.

7.1.1 AUC as an Evaluation Metric

We use the area under the receiver operating characteristic curve, or AUC-ROC (and herein referred to as just AUC). Formally, the AUC is defined as the area under the curve formed by plotting the true positive rate of a classifier as a function of its false positive rate.

7.2 Evaluation of Linear Models

In this section we focus on the evaluation of the two linear models described in 4. For our highly performant windowed linear model, in addition to performance, we are also interested in introspecting into the features prioritized by these linear models – as they are relatively simple, we are able to directly evaluate what is important to making predictions, and thereby get a qualitative sense of the relationships one would draw between clinical concepts and outcomes

7.2.1 Basic Linear Model

The basic linear model has the primary role of serving as a baseline for comparison. There is but a single parameter value to be chosen, the regularization strength λ. Following the example of prior work, we choose a single value for $\lambda = 10$ that gives good performance. Note that in this case, we do not tune λ as part of the model training, instead holding it at a fixed predetermined value.

The basic linear model used a prediction date of Jan 1, 2017, and patients were

Hyperparameter	Possible Values
λ	$\{0.05, 0.5, 5, 50\}$
$\mathcal{W}_c$	$[T_A - t', T_A]$ for all $t' \in \{15, 30, 60, 90, 180, 360, 540, 720, \infty\}$ days
α	$\{0, 0.05, 0.1, 0.15\}$

Table 7.1: Hyperparameters for Linear and Deep Models

split into training, validation, and test sets of size 97274, 5000, and 19319 respectively. The logistic regression model was fit using the SAGA solver as implemented in Scikit-Learn [16, 40]. The reported AUC was evaluated on the test set.

While simplistic, the basic linear model does have some predictive power, as shown in the row entitled Basic Linear Baseline in Table 7.3.1.

7.2.2 Windowed Linear Model

We tested the windowed linear model as described in Section 4.2 on our example prediction tasks, and found an optimal window set consisting of the intervals starting $30, 180, 365, 730$ and ∞ days before the prediction date respectively, and all ending on the prediction date itself.

In all experiments, the windowed linear models used a prediction date of Jan 1, 2017, and as with the basic linear model, patients were split into training, validation, and test sets of size 97274, 5000, and 19319 respectively. The logistic regression model was fit using the SAGA solver as implemented in Scikit-Learn [16, 40], and we tuned the hyperparameter λ to select the best value in the hyper-parameter search space of $\{0.05, 0.5, 5, 50\}$, as in Table 7.2.2 that maximized validation set AUC. The reported AUC, precision and recall were evaluated on the test set, and both precision and recall were evaluated at a threshold of 0.5.

7.2.2.1 Windowed Linear Models run on Subsets of Codes

We first consider several versions of the windowed linear model for End of Life, in which we restricted the codes to certain subsets defined via the hierarchy of the OMOP CDM. We find that many of these subsets of codes are quite predictive even in isolation, which indicates some degree of redundancy in the data. Indeed, a code

for a very specific medicine (for example, `Metformin`) will almost certainly imply that the patient is also coded for the relevant disease (in this example, `Type II Diabetes Mellitus`). We display the results of the windowed model run on all categories of data in Table 7.2. We further list the top features from each of these models – note that we remove the window of length ∞ in these cases in order to better understand the time dependencies of different codes.

A closer look at the last two columns of Table 7.2, which represent highly weighted positive and negative features, reveals that several top-weighted features across different classes of codes are effectively redundant. For example, the presence of a `Other screening mammogram` is highly correlated with `gender` (which, as per the OMOP convention, is set to 1 if the patient is female), and both codes act as highly-weighted features with negative weight. Another less subtle example is the high positive weight given to the condition code for `Antineoplastic Chemotherapy` and the clearly related procedure code for `Chemotherapy Administration`, both of which refer to the same treatment option for severe cancers. From this analysis, we chose to exclude demographic information due to redundancy, as well as not including medical device usage as these features were not at all predictive. Device codes also were excluded as they were very noisy, and mostly reflected the large amounts of generic medical apparatus needed during many hospital visits – that these features are not immediately useful is shown by the fact that they have no predictive power at all when used alone.

7.2.2.2 Windowed Linear Model Performance

Using the insights from tuning the windowed linear model on the End of Life task, we constructed and ran equivalent models for the Likelihood of Hospitalization and Surgical Procedure predictive tasks as well. Note that the hyper-parameters were chosen using a validation set of patients. We present these results in Table 7.3.1. Notably, we find that performance is uniformly and significantly better than that of the basic linear model. In addition to showing the strength of this model, this fact also shows that the basic linear model, despite its use in the literature, may not

Table 7.2: Windowed linear models trained on subsets of codes for the EoL task

Feature Type	Num. Features	AUC	Precision	Recall	Top Features (+)	Top Features (-)
Demographics	2	0.723	0.0346	0.6975	age, 0.0869	gender, -0.1821
Conditions	58475	0.811	0.0659	0.6614	30 days - Encounter for antineoplastic chemotherapy, 1.064 720 days - Antineoplastic chemotherapy, 0.5418 720 days - Alzheimer's disease, 0.5169	720 days - Other screening mammogram, -0.6411 720 days - Impotence of organic origin, -0.5571 720 days - Screening for malignant neoplasms of prostate, -0.376 720 days - Encounter for screening mammogram for malignant neoplasm of breast, -0.3108
Drugs	18378	0.720	0.0475	0.5282	720 days - Furosemide 40 MG Oral Tablet, 0.826 720 days - Dexamethasone phosphate 10 MG/ML Injectable Solution, 0.7435 720 days - 1.7 ML denosumab 70 MG/ML Injection [Xgeva], 0.647	720 days - Ibuprofen 800 MG Oral Tablet, -0.6323 720 days - Ibuprofen 600 MG Oral Tablet, -0.4342 180 days - Losartan Potassium 100 MG Oral Tablet, -0.3683
Devices	1704	0.498	0.0221	0.0564	720 days - ACCU-CHEK SAFE-T-PRO 23G LANCT, 1.899 30 days - PEN NEEDLES 8MM 31G, 1.6345 30 days - TRUEPLUS SYR 0.5ML 29GX1/2", 1.579	device 24 - ACCU-CHEK SMARTVIEW STRIP, -1.4952 device 6 - CONTOUR METER, -1.3677 device 24 - FREESTYLE 28G LANCETS, -1.1793
Procedures	29083	0.7974	0.0594	0.6569	180 days - Chemotherapy administration, intravenous infusion technique; up to 1 hour, single or initial substance/drug, 0.6028 720 days - Positron emission tomography (PET) with concurrently acquired computed tomography (CT) for attenuation correction and anatomical localization imaging; skull base to mid-thigh, 0.4224	procedure 24 - Screening mammography, bilateral (2-view study of each breast), including computer-aided detection (CAD) when performed, -0.4903 procedure 24 - Anesthesia for lower intestinal endoscopic procedures, endoscope introduced distal to duodenum, -0.4045
Specialty	398	0.7773	0.0489	0.6455	30 days - Medical Oncology, 0.5831 180 days - Hematology/Oncology, 0.5688	specialty 12 - Chiropractic, -0.7822 specialty 24 - Obstetrics/Gynecology, -0.7777 specialty 24 - Certified Clinical Nurse Specialist, -0.3089

always be a good choice for comparison. Indeed, we find that it is not necessary to use a deep model to outperform this baseline.

7.3 Evaluation of Deep Models

As in the case of the linear model, we train deep models using a prediction date of Jan 1, 2017. Patients were again split into training, validation, and test sets of size 97274, 5000, and 19319. By choosing splits identical to those used in training the linear models, direct patient-by-patient comparison is made possible, allowing for case studies to determine how and why our best-performing deep models can make superior predictions. We train the SARD architectures described in Chapter 5. In addition, we train RETAIN [13, 26], a deep learning model that previously achieved state-of-the-art performance on similar tasks. This model was chosen as a baseline as it was developed for a longitudinal EHR dataset similar to ours, achieves good performance on long-term tasks, and offers an alternative way to use attention mechanisms to ingest longitudinal health data.

Notably, we train SARD using the reverse distillation method outlined in Chapter 6, except for cases in which we intentionally train SARD to simply minimize a cross-entropy loss for comparative purposes. In this formulation, we use the windowed linear model with tuned hyper-parameters as the base model to which we fit SARD, allowing us to effectively initialize SARD to mimic an already performant, somewhat longitudinal model.

We train using a single NVIDIA k80 GPU. Our algorithms are implemented in Python 3.6 and use the PyTorch autograd library [39]. We train our deep models using an ADAM optimizer [24] with the hyperparameter settings of $\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 10^{-9}$ and a learning rate of $\eta = 2 \times 10^{-4}$. A batch size of 500 patients was used for ADAM updates. Note that a batch size of 500 did not actually fit into our GPU memory – as such, we used the largest batch size we could for each model, and simply accumulated gradients until 500 patients had their outcomes predicted, at which point we would invoke the ADAM algorithm to update the network weights.

Model \ Task Name	EoL	Surgery	LoH
Basic Linear Baseline [13]	76.3	69.9	65.4
RETAIN Recurrent Baseline [13]	82.3	80.5	72.3
Windowed Linear Baseline [43]	81.9	79.3	73.3
SARD (without RD)	81.8	81.2	70.8
SARD + RD	**85.4**	**82.8**	**74.2**

Table 7.3: AUC-ROC Scores on Test Set. + RD indicates that reverse distillation is used to train models in the indicated row.

Hyperparameters of the SARD model were chosen from the options outlined in Table 7.2.2.

7.3.1 Performance of SARD on Selected Prediction Tasks

We first present our main results, which quantify how well SARD performs on the three tasks developed in Section 7.1. As seen in Table 7.3.1, our best SARD model outperforms all baselines for each of the example tasks. Increases in AUC-ROC are significant versus the closest baseline in all cases, as measured by a paired z-test at a significance level of $p = 0.005$ [17]. In Section 7.3.4, we further explore the nuances of how SARD extracts clinical narratives, and qualitatively find that SARD is able to use a patient's entire medical history to contextualize visits, whereas the high-performing linear models seem less able to make these connections.

7.3.2 Ablation Studies of the SARD Model

We empirically test the design decisions made in Chapter 5 via ablation studies, in which we subsititute or remove different components of the architecture and evaluate performance. Each of the three primary components of the SARD architecture – the visit embedding, temporal embedding, and the self-attention mechanism – were ablated. Our time embedding is shown to be as good as if not better than alternatives such as learning embeddings for each unique visit timestamp and allowing the vector ω of frequencies to be a learned parameter of the model, and we find that carefully constructing visit representations allows us to overcome computational bar-

riers to achieving better performance. Our ablation studies also show that reverse distillation is the key driver in SARD's performance gains. Indeed, the smallest difference in ablated performance was observed when SARD's self-attention architecture was replaced with a recurrent equivalent, but reverse distillation was still used for pretraining, indicating at reverse distillation's universal applicability.

7.3.2.1 Temporal Embedding

We utilized a fixed vector of frequencies to generate our sinusoidal time embeddings, in line with past work [49] in self-attention, which found that learned time embeddings were not significantly more performant. We confirm this by swapping out the fixed-frequency time embedding for several alternative options and confirming that performance does not increase. We define these alternatives as follows:

- Learned Sinusoidal Embeddings: we set

$$\tau\left(V_j^i\right) = \mathbf{a} \odot sin\left(t_j^i \omega + \rho\right) || \ \mathbf{a} \odot cos\left(t_j^i \omega + \rho\right), \tag{7.1}$$

  where $\mathbf{a}$ is a learned vector of amplitudes, ω a learned vector of frequencies and ρ a learned vector of phases. Note that the operator $\odot$ denotes element-wise multiplication. We find that this flexibility is not necessary to improve performance when compared to an embedding using a fixed choice of frequencies, unit amplitudes and zero phases.

- Learned Embeddings: We treat each unique value of t_j^i as an independent token, and embed this token using a trained embedding matrix.

- No Time Embeddings: We set $\tau\left(V_j^i\right) = 0$. The fact that performance does not completely degrade here indicates that a combination of non-temporal features and the relative ordering of visits is enough to achieve decent performance, but not as performant as fully incorporating temporal data.

Note that reverse distillation is used in all cases of temporal embedding ablation.

7.3.2.2 Concept Embedding

The most distinctive feature of our concept embedding is the explicit aggregation of codes into visits. While this type of aggregation is standard [11, 13], it is not necessarily justified. Indeed, if a visit, or a group of codes occurring at the same time, was an important structure, that structure could be learned. We note in particular that a self-attention-based approach has merit here since we can give multiple codes identical temporal embeddings, allowing the importance of a visit to be learned.

As such, we perform an ablation study in which we feed in each code as its own separate visit into SARD. We note that the runtime of self-attention scales quadratically with the length of the input used, since every pair of input elements must attend to each other. As such, for computational reasons we must maintain a constraint that only a fixed number of visits (or, in the case of the ablated model, codes) can be inputs to the model. To accomplish this, we limit ourselves to the past one year of data, and limit each patient to 512 codes, which captures all codes over the past year for over 94% of patients. The performance of this model is found in the rows labelled `Code Level Self-Attention + RD` (when trained with reverse distillation) and `Code Level Self-Attention (without RD)` (when trained without reverse distillation) in table 7.3.2.3.

To perform a fair ablation study, we retrain SARD while using only one year's worth of data. Results from this limited model are presented in the row `SARD with 1 year's data + RD` in table 7.3.2.3 – we also present results of a visit-level self-attention model trained from without reverse distillation in row `SARD with 1 year's data (without RD)`. Our study reveals that the most performant model varies by task. However, both models fail to beat SARD trained on all available data. As such, we find a computational impetus to aggregate over visits in order to use more data in an efficient way.

7.3.2.3 Self-attention

A novel aspect of our work is its use of a self-attention architecture as a tool to ingest time-series of embedded clinical data. In the past, RNN-based approaches [11, 13, 30] have been the state-of-the-art, and as such we developed an ablation study in which we replace our architecture with a recurrent GRU-cell network, leaving the rest of the network unchanged. In table 7.3.2.3, the row `RNN (without RD)` corresponds to this ablated model trained from a random initialization, and `RNN + RD` to the ablated model trained using the same reverse distillation procedure used in SARD + RD. To ensure that our ablation fairly compared recurrent and self-attention based approaches, we preserved all other architectural elements including the visit-level input embeddings, use of temporal embeddings (fixed-frequency sinusoidal time embeddings led to the best performance), and the prediction head to aggregate the final visit representations, which here operates on the hidden states of each element of the last layer of the RNN. We found the prediction head's aggregation to be more performant and serve as a more apt comparison than the standard recurrent technique of simply predicting from the hidden state of the last element of the last layer of the RNN. This design choice helps mitigate the fact that older visits may be 'forgotten' by the RNN, by allowing these visits to directly influence the inputs of the prediction head. We find that the self-attention architecture is competitive with the RNN, so long as the RNN is also trained with reverse distillation. An important finding is that reverse distillation can also be used to successfully train highly-performant recurrent models, further validating the usefulness of this method and indicating that it can be used more generally.

7.3.3 Analysis of Generalization with Reverse Distillation

We empirically validate that the SARD model for the End of Life task after reverse distillation (but before fine-tuning) generalizes in the same way as a linear model by analyzing the predictions made by both models on a held-out validation set. As seen in Figure 7-3, we find a Spearman correlation of 0.86 between the logit outputs of

Design Choice \ Task Name	EoL	Surgery	LoH
SARD + RD	85.4	82.8	74.2
Learned Sinusoidal Embeddings	84.3	82.8	74.1
Learned Embeddings	85.0	82.3	73.7
No Time Embeddings	83.5	82.3	73.2
RNN + RD	85.5	83.0	74.0
RNN (without RD)	84.3	81.3	72.1
SARD with 1 year's data + RD	82.1	79.1	73.4
SARD with 1 year's data (without RD)	81.3	76.9	72.5
Code Level Self-Attention + RD	83.3	74.9	73.2
Code Level Self-Attention (without RD)	81.7	74.0	69.7

Table 7.4: Ablation Study Results. + RD indicates that reverse distillation is used for pretraining

the two models on held-out data, where the logit corresponding to a probability p is the natural logarithm of the odds ratio $\log(p/(1-p))$. This indicates that even for unseen patients, the models make similar predictions. Thus, the reverse-distilled deep model does indeed mimic the linear model, not just memorize its outputs at certain points.

7.3.4 Using SARD's Interpretability Technique for Case Studies

In this section we utilize the algorithm outlined in Section 5.2.1 to break down and understand the prediction made by the SARD model for the End of Life task on a specific patient of interest.

We seek to analyze a case where SARD is able to make a correct prediction but our best linear model is not. In order to convert the soft predictions of SARD and our windowed linear model to binarized predictions of outcomes, we chose decision thresholds for both models to ensure a false positive rate of 0.25 on the validation set – this resulted in a threshold of 0.493 and 0.334 for SARD and the linear model respectively. In practice, the selection of a threshold or the use of the output scores as rankings would be driven by varied downstream applications.

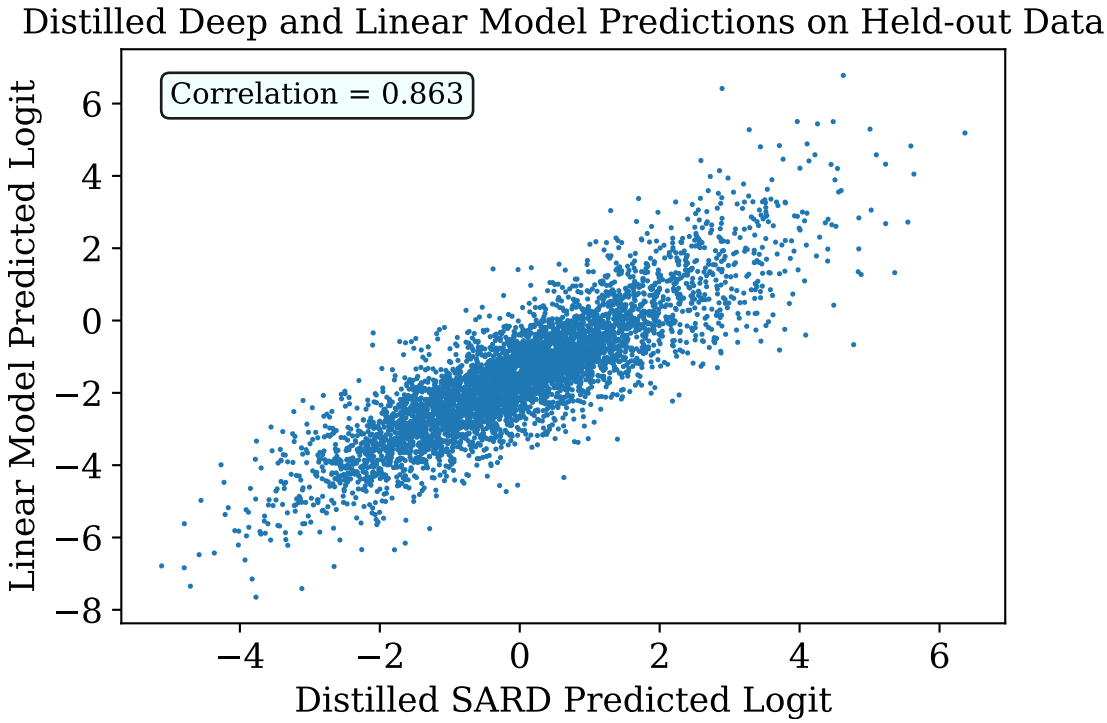

Figure 7-3: Comparison of Predictions on Held-out Data by Reverse Distilled and Linear Models

Note that in all subsequent discussion of specific cases, we reveal only years when discussing events related to individual patients, so as to censor protected health information (PHI). Our models and analyses, however, utilize the exact days on which events occur.

We consider a female patient who died in 2017 – at the time of her death she was at least 90 years old. At the thresholds determined earlier in this section, this event was correctly predicted by SARD (probability of 54.5%) but not by our linear baseline (probability of 5.4%). The patient had an extensive medical history, with over 700 recorded medical visits. To better understand why SARD accurately predicts her death while logistic regression does not, we introspect on which visits are most influential in the prediction head of the model, and how those visits were benefited by the self-attention architecture.

The specific visit that maximizes the summed SHAP score for visit importance outlined in Section 5.2.1, which we denote as the *top visit*, occurred in 2011, more than

six years prior to her death. During this visit, she experienced an `acute myocardial infarction`, `atrial fibrillation`, `chronic pulmonary heart disease`, `acute subendocardial infarction`, `congestive heart failure`, `acute on chronic systolic heart failure`, and `acquired hypothyroidism`, among other conditions. She further underwent a `coronary angiography`, in which a catheter is placed in coronary arteries to search for blood clots. See Table 7.5 for further details of this visit.

While these conditions and procedures may sound alarming, they are not highly weighted by the linear model. Aligning with medical intuition that the long-term survival rate of patients who suffer a myocardial infarction is highly dependent upon other risk factors [33], the linear model's top weighted negative features are `Insertion and placement of flow directed catheter (Swan-Ganz)`, a procedure used diagnostically to determine and eliminate risks after a myocardial infarction, and `carvedilol`, a drug known to reduce risk of death after myocardial infarction, both over length-∞ windows.

The SARD model, by contrast, is able to leverage important contextual information from throughout the patient's history thanks to its self-attention mechanism. Though the top visit occurred in 2011, SARD connects that visit with continued, albeit more minor, cardiovascular issues closer to the patient's death and understands that the patient was still at high cardiovascular risk at the time of prediction. For example, in the first layer of the first self-attention head, the three visits most strongly attended to by the top visit occur in 2016, close to the time of prediction, as visualized in Figure 7-4. During these 2016 visits, the patient is administered `clopidogrel`, a blood thinner used to prevent heart attacks and strokes in people with peripheral vascular disease, and is treated for `ankle ulcers due to atherosclerosis`. The next most strongly attended visit occurs in 2013 and includes treatment for an `ankle ulcer` and `chronic peripheral venous hypertension`. Similarly, in the first layer of the second self-attention head, the 2011 visit attends most highly to visits from 2011-2013 in which drugs for vascular diseases were administered. Importantly, when we inspect the visits *least* attended to by the top visit, we see very little cardiovascu-

Table 7.5: The 4 most predictive visits for the case study patient, by SHAP score (continued in Table 7.6)

SHAP Cumulative Score	Year of Visit	Content of Visit
0.723	2011	138384 - condition - Acquired hypothyroidism 199075 - condition - Neurogenic bladder 200174 - condition - Disorder of skin and/or subcutaneous tissue 2514405 - procedure - Initial hospital care, per day, for the evaluation and management of a patient, which requires these 3 key components: A comprehensive history; A comprehensive examination; and Medical decision making of moderate complexity... 2514421 - procedure - Inpatient consultation for a new or established patient, which requires these 3 key components: An expanded problem focused history; An expanded problem focused examination; and Straightforward medical decision making... 2514441 - procedure - Critical care, evaluation and management of the critically ill or critically injured patient; first 30-74 minutes 312327 - condition - Acute myocardial infarction 313217 - condition - Atrial fibrillation 313324 - condition - Cheyne-Stokes respiration 314054 - condition - Aortic valve disorder 315831 - condition - Chronic pulmonary heart disease 318800 - condition - Gastroesophageal reflux disease 319835 - condition - Congestive heart failure 320128 - condition - Essential hypertension 40480602 - condition - Acute on chronic systolic heart failure 40756944 - procedure - Catheter placement in coronary artery(s) for coronary angiography, including intraprocedural injection(s) for coronary angiography, imaging supervision and interpretation; with right and left heart catheterization including intraprocedural injection(s) 4148375 - procedure - Catheterization of both left and right heart 4171675 - procedure - Coronary arteriography using two catheters 432867 - condition - Hyperlipidemia 444406 - condition - Acute subendocardial infarction 77670 - condition - Chest pain 80502 - condition - Osteoporosis 81902 - condition - Urinary tract infectious disease
0.586	2011	2001449 - procedure - Open and other replacement of aortic valve with tissue graft 2100873 - procedure - Anesthesia for direct coronary artery bypass grafting; with pump oxygenator 2107121 - procedure - Replacement, aortic valve, open, with cardiopulmonary bypass; with prosthetic valve other than homograft or stentless valve 2108261 - procedure - Arterial catheterization or cannulation for sampling, monitoring or transfusion (separate procedure); percutaneous 2211359 - procedure - Radiologic examination, chest; single view, frontal 2213283 - procedure - Level IV - Surgical pathology, gross and microscopic examination Abortion - spontaneous/missed Artery, biopsy Bone marrow, biopsy Bone exostosis Brain/meninges, other than for tumor resection Breast, biopsy, not requiring microscopic evaluation 2213286 - procedure - Decalcification procedure (List separately in addition to code for surgical pathology examination) 2313886 - procedure - Insertion and placement of flow directed catheter (eg, Swan-Ganz) for monitoring purposes 2414365 - procedure - Anesthesia for patient of extreme age, younger than 1 year and older than 70 (List separately in addition to code for primary anesthesia procedure) 313217 - condition - Atrial fibrillation 314054 - condition - Aortic valve disorder 320128 - condition - Essential hypertension 40481919 - condition - Coronary atherosclerosis 4057277 - procedure - Operative external blood circulation 4125928 - procedure - Packed blood cell transfusion 4230911 - procedure - Echocardiography 77670 - condition - Chest pain

lar activity. See Tables 7.7 and 7.8 for the full contents of the visits given the most and least attention by the top visit. These persistent and recent manifestations of the patient's underlying cardiovascular disease provide context for the 2011 visit and augment its relevance at prediction time.

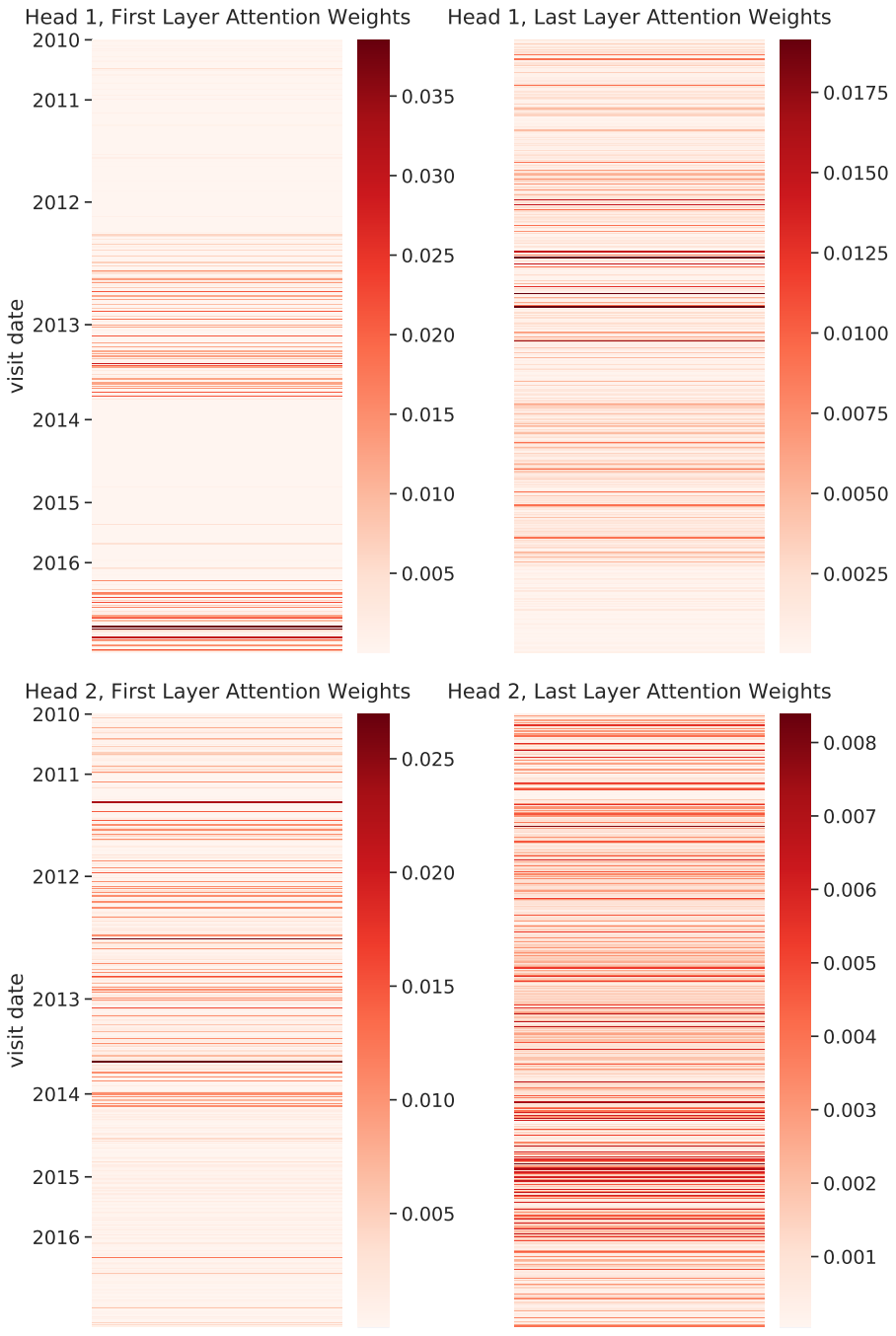

Figure 7-4: Attention weights for the case study patient's 'top visit.' While the top visit occurred in 2011, it pulls context from visits throughout the patient's history. Each panel contains a row for each of the patient's 512 visits, colored by how much attention it is given by the top visit. Notably, when we examined the visits most highly attended to in the first layer of the first self-attention head (top left panel), we noticed that they contain more recent manifestations of the same underlying atherosclerotic vascular disease present in the top visit.

Table 7.6: The 4 most predictive visits for the case study patient, by SHAP score (continued from Table 7.5).

SHAP Cumulative Score	Year of Visit	Content of Visit
0.539	2011	138384 - condition - Acquired hypothyroidism 199075 - condition - Neurogenic bladder 201069 - condition - Peptic ulcer without hemorrhage, without perforation AND without obstruction 2211359 - procedure - Radiologic examination, chest; single view, frontal 2514406 - procedure - Initial hospital care, per day, for the evaluation and management of a patient, which requires these 3 key components: A comprehensive history; A comprehensive examination; and Medical decision making of high complexity... 2514424 - procedure - Inpatient consultation for a new or established patient, which requires these 3 key components: A comprehensive history; A comprehensive examination; and Medical decision making of high complexity. Counseling and/or coordination of care with other physician 2514437 - procedure - Emergency department visit for the evaluation and management of a patient, which requires these 3 key components within the constraints imposed by the urgency of the patient's clinical condition and/or mental status: : A detailed history; A detailed examination; and Medical decision making of moderate complexity 312437 - condition - Dyspnea 313217 - condition - Atrial fibrillation 314054 - condition - Aortic valve disorder 316822 - condition - Heart murmur 317002 - condition - Low blood pressure 319034 - condition - Hypertensive heart disease without congestive heart failure 40481919 - condition - Coronary atherosclerosis 40483287 - condition - Disorder of kidney and/or ureter 4171556 - condition - Ankle ulcer 42872402 - condition - Coronary arteriosclerosis in native artery 433316 - condition - Dizziness and giddiness 434376 - condition - Acute myocardial infarction of anterior wall 438398 - condition - Leukocytosis 444070 - condition - Tachycardia 77670 - condition - Chest pain 80502 - condition - Osteoporosis
0.538	2016	197304 - condition - Ulcer of lower extremity 2314302 - procedure - Debridement (eg, high pressure waterjet with/without suction, sharp selective debridement with scissors, scalpel and forceps), open wound, (eg, fibrin, devitalized epidermis and/or dermis, exudate, debris, biofilm), including topical application(s) 2314303 - procedure - Debridement (eg, high pressure waterjet with/without suction, sharp selective debridement with scissors, scalpel and forceps), open wound, (eg, fibrin, devitalized epidermis and/or dermis, exudate, debris, biofilm), including topical application(s) 38004447 - specialty - General Surgery 4177703 - condition - Ulcer 43020432 - condition - Atherosclerosis of native arteries of the extremities 46270348 - condition - Ulcer of ankle due to atherosclerosis of native artery of limb

Table 7.7: 10 visits most highly attended by the top visit (2011); first layer, first self-attention head

Attention from Top Visit	Year of Visit	Content of Visit
0.039	2016	19075601 - drug - clopidogrel 75 MG Oral Tablet 40166135 - drug - 24 HR Oxybutynin chloride 5 MG Extended Release Oral Tablet
0.038	2016	197304 - condition - Ulcer of lower extremity 2314302 - procedure - Debridement (eg, high pressure waterjet with/without suction, sharp selective debridement with scissors, scalpel and forceps), open wound, (eg, fibrin, devitalized epidermis and/or dermis, exudate, debris, biofilm), including topical application(s), wound 38004447 - specialty - General Surgery 4177703 - condition - Ulcer 43020432 - condition - Atherosclerosis of native arteries of the extremities 46270348 - condition - Ulcer of ankle due to atherosclerosis of native artery of limb
0.030	2016	197304 - condition - Ulcer of lower extremity 2101925 - procedure - Debridement, subcutaneous tissue (includes epidermis and dermis, if performed); first 20 sq cm or less 2314302 - procedure - Debridement (eg, high pressure waterjet with/without suction, sharp selective debridement with scissors, scalpel and forceps), open wound, (eg, fibrin, devitalized epidermis and/or dermis, exudate, debris, biofilm), including topical application(s), wound 38004447 - specialty - General Surgery 4177703 - condition - Ulcer 43020432 - condition - Atherosclerosis of native arteries of the extremities 46270348 - condition - Ulcer of ankle due to atherosclerosis of native artery of limb
0.029	2013	193326 - condition - Urge incontinence of urine 2314302 - procedure - Debridement (eg, high pressure waterjet with/without suction, sharp selective debridement with scissors, scalpel and forceps), open wound, (eg, fibrin, devitalized epidermis and/or dermis, exudate, debris, biofilm), including topical application(s), wound 2414398 - procedure - Office or other outpatient visit for the evaluation and management of an established patient, which requires at least 2 of these 3 key components: A detailed history; A detailed examination; Medical decision making of moderate complexity. Counseling and/o 4171556 - condition - Ankle ulcer 4313767 - condition - Chronic peripheral venous hypertension
0.023	2016	197304 - condition - Ulcer of lower extremity 2314302 - procedure - Debridement (eg, high pressure waterjet with/without suction, sharp selective debridement with scissors, scalpel and forceps), open wound, (eg, fibrin, devitalized epidermis and/or dermis, exudate, debris, biofilm), including topical application(s), wound 2314303 - procedure - Debridement (eg, high pressure waterjet with/without suction, sharp selective debridement with scissors, scalpel and forceps), open wound, (eg, fibrin, devitalized epidermis and/or dermis, exudate, debris, biofilm), including topical application(s), wound 4177703 - condition - Ulcer 43020432 - condition - Atherosclerosis of native arteries of the extremities 46270348 - condition - Ulcer of ankle due to atherosclerosis of native artery of limb
0.023	2016	2314302 - procedure - Debridement (eg, high pressure waterjet with/without suction, sharp selective debridement with scissors, scalpel and forceps), open wound, (eg, fibrin, devitalized epidermis and/or dermis, exudate, debris, biofilm), including topical application(s), wound 2314303 - procedure - Debridement (eg, high pressure waterjet with/without suction, sharp selective debridement with scissors, scalpel and forceps), open wound, (eg, fibrin, devitalized epidermis and/or dermis, exudate, debris, biofilm), including topical application(s), wound 321596 - condition - Peripheral venous insufficiency 38004447 - specialty - General Surgery
0.022	2016	2314302 - procedure - Debridement (eg, high pressure waterjet with/without suction, sharp selective debridement with scissors, scalpel and forceps), open wound, (eg, fibrin, devitalized epidermis and/or dermis, exudate, debris, biofilm), including topical application(s), wound 321596 - condition - Peripheral venous insufficiency 38004447 - specialty - General Surgery
0.022	2016	197304 - condition - Ulcer of lower extremity 2314302 - procedure - Debridement (eg, high pressure waterjet with/without suction, sharp selective debridement with scissors, scalpel and forceps), open wound, (eg, fibrin, devitalized epidermis and/or dermis, exudate, debris, biofilm), including topical application(s), wound 2314303 - procedure - Debridement (eg, high pressure waterjet with/without suction, sharp selective debridement with scissors, scalpel and forceps), open wound, (eg, fibrin, devitalized epidermis and/or dermis, exudate, debris, biofilm), including topical application(s), wound 38004447 - specialty - General Surgery 4177703 - condition - Ulcer 43020432 - condition - Atherosclerosis of native arteries of the extremities 46270348 - condition - Ulcer of ankle due to atherosclerosis of native artery of limb
0.021	2013	2314302 - procedure - Debridement (eg, high pressure waterjet with/without suction, sharp selective debridement with scissors, scalpel and forceps), open wound, (eg, fibrin, devitalized epidermis and/or dermis, exudate, debris, biofilm), including topical application(s), wound 4171556 - condition - Ankle ulcer 4313767 - condition - Chronic peripheral venous hypertension
0.021	2012	2314302 - procedure - Debridement (eg, high pressure waterjet with/without suction, sharp selective debridement with scissors, scalpel and forceps), open wound, (eg, fibrin, devitalized epidermis and/or dermis, exudate, debris, biofilm), including topical application(s), wound 4171556 - condition - Ankle ulcer 4313767 - condition - Chronic peripheral venous hypertension

Table 7.8: 10 visits *least* attended by the top visit (2011); first layer, first self-attention head

Attention from Top Visit	Year of Visit	Content of Visit
4.62E-08	2013	19128020 - drug - {6 (Azithromycin 250 MG Oral Tablet) } Pack
5.08E-08	2014	138384 - condition - Acquired hypothyroidism 194133 - condition - Low back pain 2211970 - procedure - Bone and/or joint imaging; whole body 2314213 - procedure - Therapeutic, prophylactic, or diagnostic injection (specify substance or drug); subcutaneous or intramuscular 2314216 - procedure - Therapeutic, prophylactic, or diagnostic injection (specify substance or drug); each additional sequential intravenous push of a new substance/drug (List separately in addition to code for primary procedure) 2314217 - procedure - Therapeutic, prophylactic, or diagnostic injection (specify substance or drug); each additional sequential intravenous push of the same substance/drug provided in a facility (List separately in addition to code for primary procedure) 2314262 - procedure - Physical therapy evaluation 2314264 - procedure - Occupational therapy evaluation 2314287 - procedure - Therapeutic procedure, 1 or more areas, each 15 minutes; gait training (includes stair climbing) 2414392 - procedure - Office or other outpatient visit for the evaluation and management of a new patient, which requires these 3 key components: A detailed history; A detailed examination; Medical decision making of low complexity. Counseling and/or coordination of care with 2414398 - procedure - Office or other outpatient visit for the evaluation and management of an established patient, which requires at least 2 of these 3 key components: A detailed history; A detailed examination; Medical decision making of moderate complexity. Counseling and/o 2514412 - procedure - Observation or inpatient hospital care, for the evaluation and management of a patient including admission and discharge on the same date, which requires these 3 key components: A comprehensive history; A comprehensive examination; and Medical decision ma 312648 - condition - Benign essential hypertension 313217 - condition - Atrial fibrillation 317898 - condition - Malignant essential hypertension 40757101 - procedure - Subsequent observation care, per day, for the evaluation and management of a patient, which requires at least 2 of these 3 key components: An expanded problem focused interval history; An expanded problem focused examination; Medical decision making of mo 80816 - condition - Degeneration of intervertebral disc 81390 - condition - Idiopathic osteoporosis
7.84E-08	2014	138384 - condition - Acquired hypothyroidism 194133 - condition - Low back pain 194526 - condition - Injury of trunk 2211397 - procedure - Radiologic examination, spine, lumbosacral; 2 or 3 views 2211414 - procedure - Magnetic resonance (eg, proton) imaging, spinal canal and contents, lumbar; without contrast material 2314215 - procedure - Therapeutic, prophylactic, or diagnostic injection (specify substance or drug); intravenous push, single or initial substance/drug 2514436 - procedure - Emergency department visit for the evaluation and management of a patient, which requires these 3 key components: A detailed history; A detailed examination; and Medical decision making of moderate complexity. Counseling and/or coordination of care with o 2514437 - procedure - Emergency department visit for the evaluation and management of a patient, which requires these 3 key components within the constraints imposed by the urgency of the patient's clinical condition and/or mental status: A comprehensive history; A comprehensi 2617452 - procedure - Hospital observation service, per hour 313217 - condition - Atrial fibrillation 320128 - condition - Essential hypertension 4171556 - condition - Ankle ulcer 437176 - condition - Late effect of accidental fall
1.22E-07	2013	2414397 - procedure - Office or other outpatient visit for the evaluation and management of an established patient, which requires at least 2 of these 3 key components: An expanded problem focused history; An expanded problem focused examination; Medical decision making of low complexity. 312648 - condition - Benign essential hypertension 313217 - condition - Atrial fibrillation 38004456 - specialty - Internal Medicine 40162494 - drug - Acetaminophen 500 MG / Hydrocodone Bitartrate 5 MG Oral Tablet 433316 - condition - Dizziness and giddiness
1.24E-07	2014	40231925 - drug - Acetaminophen 325 MG / Oxycodone Hydrochloride 5 MG Oral Tablet
1.84E-07	2011	1539407 - drug - Simvastatin 40 MG Oral Tablet 19075380 - drug - Ciprofloxacin 500 MG Oral Tablet 40162494 - drug - Acetaminophen 500 MG / Hydrocodone Bitartrate 5 MG Oral Tablet
1.95E-07	2011	132466 - condition - Lumbar sprain 2414398 - procedure - Office or other outpatient visit for the evaluation and management of an established patient, which requires at least 2 of these 3 key components: A detailed history; A detailed examination; Medical decision making of moderate complexity. Counseling and/o 434123 - condition - Primary localized osteoarthrosis of pelvic region 81390 - condition - Idiopathic osteoporosis
1.99E-07	2010	134736 - condition - Backache 2211393 - procedure - Radiologic examination, spine; thoracic, 3 views 2211398 - procedure - Radiologic examination, spine, lumbosacral; minimum of 4 views 316535 - condition - Closed fracture of thoracic vertebra without spinal cord injury
2.26E-07	2014	40162515 - drug - Acetaminophen 325 MG / Hydrocodone Bitartrate 5 MG Oral Tablet 40231925 - drug - Acetaminophen 325 MG / Oxycodone Hydrochloride 5 MG Oral Table
2.45E-07	2012	40162494 - drug - Acetaminophen 500 MG / Hydrocodone Bitartrate 5 MG Oral Tablet
2.45E-07	2012	40162494 - drug - Acetaminophen 500 MG / Hydrocodone Bitartrate 5 MG Oral Tablet

Chapter 8

Conclusions and Future Work

Fundamental advances in deep learning allow for large performance increases in many domains, but in healthcare, state-of-the-art models still often rely on rule-based heuristics and feature-engineering. We present a deep learning solution to the problem of making predictions using longitudinal health data: SARD, a self-attention based architecture that extracts contextual information across timelines of medical events. Significant performance gains using this model are underpinned by the use of reverse distillation, our novel pre-training procedure which demonstrably allows a deep model to be initialized to mimic a simpler but performant baseline. Using these innovations, we are able to exceed state-of-the-art performance for several medical prediction tasks. To our knowledge, SARD is the first successful adaptation of the self-attention paradigm to structured longitudinal health data. In order to run SARD on standardized OMOP CDM data, we also develop Prediction Library, a software pipeline to efficiently manipulate longitudinal health data for deep learning.

8.1 Broader Impact

Machine learning as applied to healthcare has the potential to greatly improve outcomes for patients. Our work in particular has the primary application of being used to determine which patients would benefit from interventions, which could obviate the need for more intensive and invasive treatments in the future. If implemented

successfully, our model would have several positive impacts in healthcare. Our results also have the potential to impact the machine learning community:

- **Improved interventions** for patients. If we are able to correctly predict when patients will suffer adverse outcomes sufficiently far into the future, clinicians will be able to intervene to help prevent or ameliorate the impact of these potential issues.

- **The extension of self-attention to clinical machine learning**, which we believe is of great interest due to the potential for performance gains and increased interpretability. SARD's success shows that such models can indeed work in practice, and can serve as a starting point for further research.

- **Combining expert domain knowledge and deep learning** through reverse distillation. Our new training method is a novel way to adapt and improve upon high-performing algorithms that rely on heuristics and data-engineering. This has the potential to inspire performance gains in healthcare, where such models are prevalent, but in other domains as well.

When work like ours is deployed in the field, it is critical to regularly determine how well the algorithms perform in general, how fairly they allocate resources, and how their predictions actually affect patient outcomes. The following considerations summarize some of the risks that must be mitigated when using SARD in a live clinical setting:

- **Equity of access** is a key goal in healthcare. Like many deep-learning paradigms, SARD performs better on some cases than others. For example, we find that it performs better on patients for whom more data is available. Such properties put groups of people for whom less data is available, or are otherwise disadvantaged in terms of model performance, at risk of less accurate predictions, and therefore less effective interventions [38]. Our future work will certainly focus on analyzing SARD and its predictions as well as downstream pipelines that ingest these predictions from the perspective of algorithmic fairness. This will help to ensure that SARD can indeed be used to help improve outcomes for all.

- **Clinical usage** of predictions from SARD must be used in a way that actually improves patient outcomes. Even when predictions are accurate and fair, they also should be used appropriately to improve care. Medical researchers have already set forth ethical and clinical guidelines governing how one should use predictions of sensitive events such as end of life [15], and in order for SARD's performance to indeed translate to improved patient outcomes it is necessary that such principles are properly followed.

8.2 Future Directions of Research

Reverse distillation is just one successful method by which self-attention based predictive models can be initialized – in natural language processing, similar architectures have been found to substantially benefit from unsupervised learning with tasks such as imputing words or predicting sentence order [14]. Now that we have demonstrated that these architectures perform competitively on longitudinal clinical data, it opens the door to similarly designing unsupervised learning tasks to improve performance even further. Furthermore, while we are able to introspect and interpret predictions made by SARD at an individual level, future work may find more global ways to interpret how visits attend to each other through the lens of medical logic, similar to how self-attention as applied to natural language has been found to replicate nuanced grammatical and linguistic phenomena [28].

Bibliography

[1] Observational health data sciences and informatics wiki. https://www.ohdsi.org/web/wiki/doku.php.

[2] Hameer Abbasi. Sparse: A more modern sparse array library. In *Proceedings of the 17th Python in Science Conference*, pages 27–30, 2018.

[3] Simon T Adams and Stephen H Leveson. Clinical prediction rules. *BMJ*, 344:d8312, 2012.

[4] Muhammad A Ahmad, Carly Eckert, Greg McKelvey, Kiyana Zolfagar, Anam Zahid, and Ankur Teredesai. Death vs. data science: predicting end of life. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[5] Anand Avati, Kenneth Jung, Stephanie Harman, Lance Downing, Andrew Ng, and Nigam H Shah. Improving palliative care with deep learning. *BMC medical informatics and decision making*, 18(4):122, 2018.

[6] Richard F Averill, Robert L Mullin, Barbara A Steinbeck, Norbert I Goldfield, and Thelma M Grant. Development of the icd-10 procedure coding system (icd-10-pcs). *Topics in health information management*, 21(3):54–88, 2001.

[7] Paul Beattie and Roger Nelson. Clinical prediction rules: what are they and what do they tell us? *Australian Journal of Physiotherapy*, 52(3):157–163, 2006.

[8] Jamie C Brehaut, Ian G Stiell, Laura Visentin, and Ian D Graham. Clinical decision rules "in the real world": how a widely disseminated rule is used in everyday practice. *Academic emergency medicine*, 12(10):948–956, 2005.

[9] Elliot G Brown, Louise Wood, and Sue Wood. The medical dictionary for regulatory activities (meddra). *Drug safety*, 20(2):109–117, 1999.

[10] 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.

[11] 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*, pages 301–318, 2016.

[12] Edward Choi, Mohammad Taha Bahadori, Le Song, Walter F Stewart, and Jimeng Sun. Gram: graph-based attention model for healthcare representation learning. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 787–795, 2017.

[13] Edward Choi, Mohammad Taha Bahadori, Jimeng Sun, Joshua Kulas, Andy Schuetz, and Walter Stewart. Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. In *Advances in Neural Information Processing Systems*, pages 3504–3512, 2016.

[14] Youngduck Choi, Chill Yi-I Chiu, and David Sontag. Learning low-dimensional representations of medical concepts. *AMIA Summits on Translational Science Proceedings*, 2016:41, 2016.

[15] W Daniel Doty and Robert M Walker. Medical futility. *Clinical cardiology*, 23(S2):6–16, 2000.

[16] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems*, pages 1646–1654, 2014.

[17] Elizabeth R DeLong, David M DeLong, and Daniel L Clarke-Pearson. Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach. *Biometrics*, pages 837–845, 1988.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[19] Centers for Medicare & Medicaid Services et al. Hcpcs general information. *Retrieved from CMS. gov*, 2014.

[20] Junyi Gao, Cao Xiao, Yasha Wang, Wen Tang, Lucas M Glass, and Jimeng Sun. Stagenet: Stage-aware neural networks for health risk prediction. In *Proceedings of The Web Conference 2020*, pages 530–540, 2020.

[21] Hrayr Harutyunyan, Hrant Khachatrian, David C Kale, Greg Ver Steeg, and Aram Galstyan. Multitask learning and benchmarking with clinical time series data. *Scientific data*, 6(1):1–18, 2019.

[22] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[23] George Hripcsak, Jon D Duke, Nigam H Shah, Christian G Reich, Vojtech Huser, Martijn J Schuemie, Marc A Suchard, Rae Woong Park, Ian Chi Kei Wong, Peter R Rijnbeek, et al. Observational health data sciences and informatics (ohdsi): opportunities for observational researchers. *Studies in health technology and informatics*, 216:574, 2015.

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Jacob Krive, Mahatkumar Patel, Lisa Gehm, Mark Mackey, Erik Kulstad, et al. The complexity and challenges of the icd-9-cm to icd-10-cm transition in emergency departments. *The American journal of emergency medicine*, 33(5):713, 2015.

[26] Bum Chul Kwon, Min-Je Choi, Joanne Taery Kim, Edward Choi, Young Bin Kim, Soonwook Kwon, Jimeng Sun, and Jaegul Choo. Retainvis: Visual analytics with interpretable and interactive recurrent neural networks on electronic medical records. *IEEE transactions on visualization and computer graphics*, 25(1):299–309, 2018.

[27] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Advances in Neural Information Processing Systems*, pages 5244–5254, 2019.

[28] Yongjie Lin, Yi Chern Tan, and Robert Frank. Open sesame: Getting inside bert's linguistic knowledge. *arXiv preprint arXiv:1906.01698*, 2019.

[29] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[30] Fenglong Ma, Radha Chitta, Jing Zhou, Quanzeng You, Tong Sun, and Jing Gao. Dipole: Diagnosis prediction in healthcare via attention-based bidirectional recurrent neural networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1903–1911, 2017.

[31] Fenglong Ma, Yaqing Wang, Houping Xiao, Ye Yuan, Radha Chitta, Jing Zhou, and Jing Gao. A general framework for diagnosis prediction via incorporating medical code descriptions. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1070–1075. IEEE, 2018.

[32] Liantao Ma, Chaohe Zhang, Yasha Wang, Wenjie Ruan, Jiangtao Wang, Wen Tang, Xinyu Ma, Xin Gao, and Junyi Gao. Concare: Personalized clinical feature embedding via capturing the healthcare context. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 833–840, 2020.

[33] Craig A Martin, PL Thompson, BK Armstrong, MS Hobbs, and Nicholas de Klerk. Long-term prognosis after recovery from myocardial infarction: a nine year follow-up of the perth coronary register. *Circulation*, 68(5):961–969, 1983.

[34] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[35] Riccardo Miotto, Li Li, Brian A Kidd, and Joel T Dudley. Deep patient: an unsupervised representation to predict the future of patients from the electronic health records. *Scientific reports*, 6(1):1–10, 2016.

[36] M Muralikrishna et al. Improved unnesting algorithms for join aggregate sql queries. In *VLDB*, volume 92, pages 91–102. Citeseer, 1992.

[37] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632, 2005.

[38] Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. Dissecting racial bias in an algorithm used to manage the health of populations. *Science*, 366(6464):447–453, 2019.

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[41] Alvin Rajkomar, Eyal Oren, Kai Chen, Andrew M Dai, Nissan Hajaj, Michaela Hardt, Peter J Liu, Xiaobing Liu, Jake Marcus, Mimi Sun, et al. Scalable and accurate deep learning with electronic health records. *NPJ Digital Medicine*, 1(1):18, 2018.

[42] Rajesh Ranganath, Adler J Perotte, Noémie Elhadad, and David M Blei. The survival filter: Joint survival analysis with a latent time series. In *UAI*, pages 742–751, 2015.

[43] Narges Razavian, Saul Blecker, Ann Marie Schmidt, Aaron Smith-McLallen, Somesh Nigam, and David Sontag. Population-level prediction of type 2 diabetes from claims data and analysis of risk factors. *Big Data*, 3(4):277–287, 2015.

[44] Narges Razavian, Jake Marcus, and David Sontag. Multi-task prediction of disease onsets from longitudinal laboratory tests. In *Machine Learning for Healthcare Conference*, pages 73–100, 2016.

[45] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[46] Jenna M Reps, Martijn J Schuemie, Marc A Suchard, Patrick B Ryan, and Peter R Rijnbeek. Design and implementation of a standardized framework to generate and evaluate patient-level prediction models using observational healthcare data. *Journal of the American Medical Informatics Association*, 25(8):969–975, 2018.

[47] Kent A Spackman, Keith E Campbell, and Roger A Côté. Snomed rt: a reference terminology for health care. In *Proceedings of the AMIA annual fall symposium*, page 640. American Medical Informatics Association, 1997.

[48] Ethan Steinberg, Ken Jung, Jason A Fries, Conor K Corbin, Stephen R Pfohl, and Nigam H Shah. Language models are an effective patient representation learning technique for electronic health record data. *arXiv preprint arXiv:2001.05295*, 2020.

[49] 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*, pages 5998–6008, 2017.

[50] Neo Wu, Bradley Green, Xue Ben, and Shawn O'Banion. Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv preprint arXiv:2001.08317*, 2020.

[51] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

[52] Fan Zhang, Tong Wu, Yunlong Wang, Yong Cai, Cao Xiao, Emily Zhao, Lucas Glass, and Jimeng Sun. Predicting treatment initiation from clinical time series data via graph-augmented time-sensitive model. *arXiv preprint arXiv:1907.01099*, 2019.