Deep Learning for Precision Medicine

Cristóbal Esteban

München 2018

Deep Learning for Precision Medicine

Cristóbal Esteban

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Cristóbal Esteban
aus Bilbao, Spanien

München, den 08.08.2018

Erstgutachter: Prof. Dr. Volker Tresp
Zweitgutachter: Prof. Dr. Bertram Müller-Myhsok
Tag der mündlichen Prüfung: 04.10.2018

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12.07.11, § 8, Abs. 2 Pkt. .5.)

Hiermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Esteban, Cristóbal

Name, Vorname

München, 16.09.2019

Ort, Datum

Unterschrift Doktorand/in

Formular 3.2

Contents

List of Figures

Acknowledgments

I would like to thank everyone that helped me during this stage of my life.

To my wife Carmen, who moved to Germany so that I could take this job. And to our daughter also named Carmen, who was not yet present during my PhD, but that doesn't really matter.

To my supervisor Volker Tresp, because he gave me the opportunity to join his group, he guided me during all these years, and provided a great balance between supervision and freedom.

To my father, mother and brother, always helpful and supportive, and who taught me that good enough is not enough. And to my grandparents, who worked hard during difficult times and thank to them I can have a job that I love doing.

To my colleagues Denis Krompaß, Stephan Baier, Yinchong Yang, Sigurd Spieckerman and Xueyan Jiang, with whom I learned a lot, shared many laughs and had a bunch of beers.

To the heads of the Machine Intelligence group at Siemens: Michael Skubacz and Ulli Waltinger who always supported my research.

To my friend Fernando Sancho, who never gets tired of helping and providing smart advice.

To every member of the MLPM Marie Curie ITN community. Special mention to Dean Bodenham, Yi Zhong, Melanie Fernández and Meiwen Jia, with whom I shared many good moments. Also to Karsten Borgwardt, who set up and lead the ITN, and to Bertram

Müller-Myhsok who agreed to be external examiner of this thesis.

To the heads of the labs that I visited during my PhD: Gunnar Rätsch from the Memorial Sloan Kettering Cancer Center, with special mention to David Kuo from his lab, who is always willing to help; Antonio Artés from University Carlos III, and Kristel van Steen from University of Liege, with special mention to Kirill Bessonov and Ramouna Fuladi from her group, which were very helpful.

To Oliver Staeck and Enrique Baca-García for the advice they provided from a medical point of view.

To everyone that reads this section and feels upset for not being in here, since I am sure I forgot to include people that deserve to be acknowledged.

Abstract

As a result of the recent trend towards digitization, an increasing amount of information is recorded in clinics and hospitals, and this increasingly overwhelms the human decision maker. This issue is one of the main reasons why Machine Learning (ML) is gaining attention in the medical domain, since ML algorithms can make use of all the available information to predict the most likely future events that will occur to each individual patient. Physicians can include these predictions in their decision processes which can lead to improved outcomes. Eventually ML can also be the basis for a decision support system that provides personalized recommendations for each individual patient.

It is also worth noticing that medical datasets are becoming both longer (i.e. we have more samples collected through time) and wider (i.e. we store more variables). Therefore we need to use ML algorithms capable of modelling complex relationships among a big number of time-evolving variables. A kind of models that can capture very complex relationships are Deep Neural Networks, which have proven to be successful in other areas of ML, like for example Language Modelling, which is a use case that has some some similarities with the medical use case.

However, the medical domain has a set of characteristics that make it an almost unique scenario: multiple events can occur at the same time, there are multiple sequences (i.e. multiple patients), each sequence has an associated set of static variables, both inputs and outputs can be a combination of different data types, etc. For these reasons we need to develop approaches specifically designed for the medical use case.

In this work we design and develop different kind of models based on Neural Networks that are suitable for modelling medical datasets. Besides, we tackle different medical tasks and datasets, showing which models work best in each case.

The first dataset we use is one collected from patients that suffered from kidney failure. The data was collected in the Charité hospital in Berlin and it is the largest data collection of its kind in Europe. Once the kidney has failed, patients face a lifelong treatment and

periodic visits to the clinic for the rest of their lives. Until the hospital finds a new kidney for the patient, he or she must attend to the clinic multiple times per week in order to receive dialysis, which is a treatment that replaces many of the functions of the kidney. After the transplant has been performed, the patient receives immunosuppressive therapy to avoid the rejection of the transplanted kidney. Patients must be periodically controlled to check the status of the kidney, adjust the treatment and take care of associated diseases, such as those that arise due to the immunosuppressive therapy. This dataset started being recorded more than 30 years ago and it is composed of more than 4000 patients that underwent a renal transplantation or are waiting for it. The database has been the basis for many studies in the past.

Our first goal with the nephrology dataset is to develop a system to predict the next events that will be recorded in the electronic medical record of each patient, and thus to develop the basis for a future clinical decision support system. Specifically, we model three aspects of the patient evolution: medication prescriptions, laboratory tests ordered and laboratory test results.

Besides, there are a set of endpoints that can happen after a transplantation and it would be very valuable for the physicians to be able to know beforehand when one of these is going to happen. Specifically, we also predict whether the patient will die, the transplant will be rejected, or the transplant will be lost. For each visit that a patient makes to the clinic, we anticipate which of those three events (if any) will occur both within 6 months and 12 months after the visit.

The second dataset that we use in this thesis is the one collected by the MEmind Wellness Tracker, which contains information related to psychiatric patients. Suicide is the second leading cause of death in the 15-29 years age group, and its prevention is one of the top public health priorities. Traditionally, psychiatric patients have been assessed by self-reports, but these suffer from recall bias. To improve data quantity and quality, the MEmind Wellness Tracker provides a mobile application that enables patients to send daily reports about their status. Thus, this application enables physicians to get information about patients in their natural environments. Therefore this dataset contains sequential information generated by the MEmind application, sequential information generated during medical visits and static information of each patient. Our goal with this dataset is to predict the suicidal ideation value that each patient will report next.

In order to model both datasets, we have developed a set of predictive Machine Learning models based on Neural Networks capable of integrating multiple sequences of data with

the background information of each patient. We compare the performance achieved by these approaches with the ones obtained with classical ML algorithms.

For the task of predicting the next events that will be observed in the nephrology dataset, we obtained the best performance with a Feedforward Neural Network containing a representation layer. On the other hand, for the tasks of endpoint prediction in nephrology patients and the task of suicidal ideation prediction, we obtained the best performance with a model that combines a Feedforward Neural Network with one or multiple Recurrent Neural Networks (RNNs) using Gated Recurrent Units. We hypothesize that this kind of models that include RNNs provide the best performance when the dataset contains long-term dependencies.

To our knowledge, our work is the first one that develops these kind of deep networks that combine both static and several sources of dynamic information. These models can be useful in many other medical datasets and even in datasets within other domains. We show some examples where our approach is successfully applied to non-medical datasets that also present multiple variables evolving in time.

Besides, we installed the endpoints prediction model as a standalone system in the Charité hospital in Berlin. For this purpose, we developed a web based user interface that the physicians can use, and an API interface that can be used to connect our predictive system with other IT systems in the hospital.

These systems can be seen as a recommender system, however they do not necessarily generate valid prescriptions. For example, for certain patient, a system can predict very high probabilities for all antibiotics in the dataset. Obviously, this patient should not take all antibiotics, but only one of them. Therefore, we need a human decision maker on top of our recommender system. In order to model this decision process, we used an architecture based on a Generative Adversarial Network (GAN). GANs are systems based on Neural Networks that make better generative models than regular Neural Networks. Thus we trained one GAN that works on top of a regular Neural Network and show how the quality of the prescriptions gets improved. We run this experiment with a synthetic dataset that we created for this purpose.

The architectures that we developed, are specially designed for modelling medical data, but they can be also useful in other use cases. We run experiments showing how we train them for modelling the readings of a sensor network and also to train a movie recommendation engine.

Chapter 1

Introduction

As a result of the recent trend towards digitization, an increasing amount of information is recorded in clinics and hospitals, and this increasingly overwhelms the human decision maker. This issue is one of the main reasons why Machine Learning (ML) is gaining attention in the medical domain, since ML algorithms can make use of all the available information to predict the most likely future events that will occur to each individual patient. Physicians can include these predictions in their decision processes which can lead to improved outcomes. Eventually ML can also be the basis for a decision support system that provides personalized recommendations for each individual patient.

However this large amounts of medical data are rarely exploited with Machine Learning algorithms. This happens due to two reasons that will be detailed later in this chapter, but they can be summarized in two facts: on one hand analyzing medical data usually requires a huge preprocessing effort, and on the other hand there is a lack of Machine Learning models tailored to suit the features of medical data. In this thesis we will contribute to improve both aspects, explaining the preprocessing steps that we took in our datasets, and presenting models specifically designed to tackle some of the challenges present in medical data.

We will work with two datasets, each of them representing a different use case. The first one was collected by the nephrology department of the Charité hospital in Berlin and contains information about patients that suffered from kidney failure. This dataset contains the full Electronic Health Record of each patient, including tests performed, medications prescribed, background information about the patients, etc. The second dataset we use is composed of psychiatric patients. It was collected by the Fundación Jimenez Díaz Hospital in Spain and the MEmind Wellness Tracker. In this dataset each patient is composed

of some static information combined with two sequences of data: one recorded by the physicians during the visits of the patient to the clinic, and another sequence reported by the patients using the web application provided by the wellness tracker. These datasets will be explained in more detail in the next chapters.

It is worth noticing that medical datasets are becoming both longer (i.e. we have more samples collected through time) and wider (i.e. we store more variables). Therefore we need to use ML algorithms capable of modelling complex relationships among a big number of time-evolving variables. A kind of models that can capture very complex relationships are Deep Artificial Neural Networks, which have proven to be successful in other areas of ML.

Artificial Neural Networks, from now Neural Networks (NNs), are a type of Machine Learning algorithm that was developed in the 80s. This thesis assumes that the reader is familiarized with this type of algorithms, and in the following paragraphs we provide a very brief introduction to NNs with references to their most basic concepts.

The main component of a NN is the Perceptron, which corresponds to a linear combination of the inputs followed by an activation function, which is usually a non-linear function such as a sigmoid or an hyperbolic function. Perceptrons can be stacked forming layers, and such layers can be also stacked forming what is known as a Multi-layer Perceptron or an Artificial Neural Network. The simplest NNs are the ones composed of a single hidden layer, which mathematically can be represented as:

$$h = \sigma\left(W_i x + b_i\right) \tag{1.1}$$

$$\hat{y} = \sigma\left(W_o h + b_o\right) \tag{1.2}$$

where x is a vector that contains the input variables, $\hat{y}$ is a vector containing the outputs of the model, W_i and W_o are matrices that contain the parameters of the input and output layers respectively, b_i and b_o are vectors that contain the bias or offset also of the input and output layers respectively and σ is a non linear function such as the sigmoid function. The intermediate variable h is a vector usually referred as "hidden layer", since its value is usually not observed by the user of the model.

The most common strategy to train NNs is the backpropagation optimization algorithm [51], which consist of propagating the gradient of the prediction error to each parameter of the network, and subsequently adapt each of them into the direction that decreases the error. These models can contain a very large amount of parameters, and therefore they

can potentially model very complex processes. There are mainly three constraints that make it hard for NNs to learn complex tasks:

- Data: The more parameters a model has, the more data it is needed to train it. Thus, in order to learn a model with many parameters such as a NN, large amounts of data are needed. Unfortunately, a theoretical framework to evaluate the amount of parameters that can be learned with certain dataset is currently missing. However, the recent trend in healthcare towards digitization is resulting in big databases that are suitable for training large NNs.

- Computing power: Fitting a large amount of parameters requires a high amount of computing power. Thanks to the rapid improvements in GPU technologies, they have become a much more affordable alternative than CPUs. Thus, GPUs enable researchers and engineers to train much more complex models.

- Vanishing gradients: As previously mentioned, the Backpropagation algorithm consists of propagating the gradient of the error until each parameter of the network. However, such gradient tend to 0 when it is passed through many of the layers that compose a NN. Thus, this mathematical constraint prevents us from training NNs composed of many layers. During the last years, researchers have developed algorithmic improvements that enable us to partially overcome this limitation in order to train large networks. Further details on this matter will be provided in Chapter 3.

The term "Deep Learning" is often associated with NNs with more than one hidden layer. State of the art models for some applications such as Computer Vision are currently composed of up to 70 layers [40]. These Deep Neural Networks have provided notable improvements with respect to other techniques in many applications such as computer vision [40] and natural language processing [61, 26].

There is a notable parallelism among the prediction of clinical events and the field of Language Modelling, where Deep Learning has also proven to be very successful [61, 8]. One could imagine that each word of a text represents an event. Therefore a text would be a stream of events and the task of Language Modelling is to predict the next event in the stream. For this reason, we can get inspired by Language Modelling to create models that predict clinical events. In the clinical setting each patient would be a stream of events, and our task would be to predict the next events that will occur.

However, the medical domain has a set of characteristics that make it an almost unique scenario. The main ones are:

- Multiple events can occur at the same time: For example, multiple medications can be prescribed simultaneously; or multiple lab results can be ordered at the same time.

- Clinical datasets are composed of multiple sequences: Each patient in the dataset is composed of, at least, one sequence of data. It is also possible that each patient is composed of multiple sequences recorded from different sources of data, with different periodicities, time resolutions, etc.

- Each patient has an associated set of static data: Aside from the sequential data, each patient is also often represented by a set of static variables containing background information such as: age, gender, blood type, etc.

- Clinical datasets are usually composed of several variable types. We can find boolean variables (i.e. medication prescriptions), categorical variables (i.e. blood type), integer variables (i.e. age) and real numbers (i.e. calcium amount in blood). Medical datasets can also contain images, free text, etc.

Therefore, it is critical to develop Machine Learning algorithms that are capable of finding complex relationships among all those sources of information in order to predict future events. For these reasons we need to develop approaches specifically designed for the medical use case. Particularly, in this thesis, we focus in developing Machine Learning models based on Neural Networks that are capable of exploiting several sources of dynamic and static information. These approaches enable us to easily integrate additional sources of information and to model complex relationships between all the variables.

Aside from these characteristics, medical datasets also present a distinct set of issues:

- Hetereogenity / inconsistencies: Medical databases are usually collected in a collaborative manner, where many physicians take part in the process. The collection process is often not properly constrained and validated, which leads to physicians recording the information in diverse ways.

- Repeated entities: Often we find elements (medications, diagnosis, etc.) that are present multiple times in the database but with slightly different names.

- Incomplete timestamps: Some medical datasets contain the date when certain event happened, but the time is missing. In these situations it is not possible to extract the exact sequence in which the events took place.

- Complex domain: Due to the complexity of the medical scenario, it is very hard for an outsider to make sense out of a database without additional explanations by the physicians. Therefore, lots of interactions with the physicians are needed to understand the data and define the specific task that can be solved with Machine Learning algorithms.

- Lots of free text: A huge amount of information is usually stored in form of free text. In order to make use of this information one has to convert the free text into structured data. Such conversion process is not trivial and often just a small portion of the information is retrieved.

- Data quality changes with time: Data quality can change over time due to improvements in the data collection process, change of instruments and machines, etc.

- Database entries not treated as immutable: Sometimes physicians are allowed to modify elements in the database, such as diagnostics, which leads to a loss of information concerning the real sequence of events that occurred.

Due to these issues, data cleaning and preprocessing is a tough and very time consuming process. It is also mandatory to work closely with the physicians that have collected the information and the IT people that designed the database. This fact may seem contradictory to one of the main goals of Artificial Intelligence, which is to make computers learn from a dataset without any human intervention. However, considering the limitations of current Machine Learning models, it is essential to include expert knowledge in healthcare data preprocessing pipelines.

The rest of the thesis is organized as follows:

In Chapter 2 we tackle the problem of predicting which events will be observed next for each patient in the Nephrology dataset. For this purpose we develop a Feedforward Neural Network that contains a representation layer. We use this model to predict the events that will be observed next in the patients. The content of this chapter is based on the following publication:

- Cristóbal Esteban, Danilo Schmidt, Denis Krompaß and Volker Tresp. (2015, October). Predicting sequences of clinical events by using a personalized temporal latent embedding model. In Healthcare Informatics (ICHI), 2015 International Conference on (pp. 130-139). IEEE.

In Chapter 3 we provide a brief explanation of Recurrent Neural Networks. We explain why it is difficult for Recurrent Neural Networks remember events that happened far in the past and we show what are the current solutions for this problem. Recurrent Neural Networks will be the main model used in this thesis.

In Chapter 4 we focus on predicting a set of endpoints for Nephrology patients. Specifically, we try to anticipate whether if in the next 6 or 12 month there will be a rejection of the kidney, the graft will be lost and/or the patient will die. For this purpose, we develop a Neural Network architecture that combines static and dynamic information using a regular Feedforward Neural Network and a Recurrent Neural Network. The content of this chapter is based on the following publication:

- Cristóbal Esteban, Oliver Staeck, Stephan Baier, Yinchong Yang, and Volker Tresp (2016, October). Predicting clinical events by combining static and dynamic information using recurrent neural networks. In Healthcare Informatics (ICHI), 2016 IEEE International Conference on (pp. 93-101). IEEE.

In Chapter 5 we try to anticipate the future suicidal ideation value of the patients in the MEmind Wellness Tracker. In this case, we extend the architecture presented in Chapter 4 in order to combine multiple sequences of data with static data. For this purpose we will include in our model multiple Recurrent Neural Networks.

This thesis is not meant to be just theoretical work, but also applied work. Thus, it was our goal to actually integrate a prototype of one of the developed models into a hospital. In Chapter 6 we show how we developed a software architecture that was deployed in the Charité hospital of Berlin.

In Chapter 7 we show how the models we presented so far can provide good recommendations but not valid decisions. We show how Generative Adversarial Networks [38] can be used to model the human decision process.

Chapter 8 shows a generalization of the presented models that can also be useful in other use cases where there is static data combined with dynamic data. The content of this chapter is based on the following publication:

- Cristóbal Esteban, Volker Tresp, Yinchong Yang, Stephan Baier, and Denis Krompaß. (2016, July). Predicting the co-evolution of event and knowledge graphs. In Information Fusion (FUSION), 2016 19th International Conference on (pp. 98-105). IEEE.

Finally in Chapter 9 we present the conclusion for this thesis.

Chapter 2

Predicting Sequences of Clinical Events by using a Personalized Temporal Latent Embedding Model

2.1 Towards Machine Learning-based Clinical Decision Support Systems

It is well known that data observed in clinical practice can lead to important insights and can complement information gathered from controlled clinical studies [80]. One argument is that data from clinical practice reflects the natural mix of patients whereas patients participating in clinical studies typically have another composition: they are carefully selected, they should not have other problems as the one under study, and they should not receive any other treatment. Also, a future personalized medicine needs to be based on many attributes from a large number of patients, information that can be collected from data recorded during the clinical practice [64, 97].

In this chapter we focus on the prediction of clinical events, such as decisions, procedures, measurements and other observations. We model the whole evolution of each individual patient, which is composed of thousands of single events. A good predictive system could have many applications, for example, as part of a decision support system that predicts common practice in a clinical setting and which could alert in case of unusual orders. Eventually, a predictive system could also be used to optimize decisions, although here, confounding variables can be a problem. If many dimensions are measured, the avail-

able information might include direct or indirect information on important confounders, alleviating the problem [81, 97].

We are addressing the issue from a "Big Data" perspective and use a large dataset collected from patients that suffered from kidney failure. The data was collected in the Charité hospital in Berlin and it is the largest data collection of its kind in Europe. Once the kidney has failed, patients face a lifelong treatment and periodic visits to the clinic for the rest of their lives. Until the hospital finds a new kidney for the patient, the patient must attend to the clinic multiple times per week in order to receive dialysis, which is a treatment that replaces many of the functions of the kidney. After the transplant has been performed, the patient receives immunosuppressive therapy to avoid the rejection of the transplanted kidney. The patient must be periodically controlled to check the status of the kidney, adjust the treatment and take care of associated diseases, such as those that arise due to the immunosuppressive therapy. The usual procedure at the Charité University Hospital of Berlin for these periodic evaluations is that the visiting patient undergoes some laboratory testing in the morning, followed by the prescription of pertinent medications in the afternoon based on the results of the test.

The dataset contains every event that happened to each patient concerning the kidney failure and all its associated events: medications prescribed, hospitalizations, diagnoses, laboratory tests, etc. [54, 82]. The dataset started being recorded more than 30 years ago and it is composed of more than 4000 patients that underwent a renal transplantation or are waiting for it. For example, the database contains more than 1200 medications that have been prescribed more than 250000 times, and the results of more than 450000 laboratory analysis. The database has been the basis for many studies in the past [42, 20, 19, 15]. In this work we study if future events for a patient can be predicted given the past events of the same patient. This is particularly important for the estimation of drug-drug interactions (DDI) and adverse drug reactions (ADR) in patients after renal transplantation.

Note that the data is extremely high-dimensional (there are thousands of diagnosis, procedures, lab results to consider) and sparse, since most combinations are unobserved. In recent years a number of approaches for this type of data situation have been developed in other application fields. These approaches are based on the concept of a low-dimensional latent embedding of the entities and events of interest in combination with Neural Network models and showed superior predictive performance in their respective domains. Examples are leading language models in natural language processing [8], the winning entries in the Netflix competition for the development of movie recommendation systems [47] and

approaches for learning in knowledge graphs [65]. A new aspect here is that the temporal sequence of events plays an important role. In this chapter we extend these models to be applicable towards temporal sequential models for the prediction of events in a clinical setting and we develop a new model that extends the Markov property of language models towards a personalized model and a long-term memory. We compare the prediction accuracy of these approaches with other leading modelling approaches such as a nearest neighbor methods, Naive Bayes classifier and Logistic Regression models.

There have been efforts within the medical domain to simultaneously predict a reduced number of events [55, 71] and also to detect patterns within a larger amount of events [100]. Our dataset consists of sequences of high-dimensional sparse data and in this situation latent embedding approaches as used in language models [8], collaborative filtering [47] and knowledge graph models [65] have been very successful. In these models, the latent embeddings represent general entities such as users, items, or simply words, and the idea is that the embeddings represent the essence of the entities in form of low-dimensional real-valued representations. Latent embeddings were introduced as a suitable strategy for clinical data in [48] by predicting hospital readmissions. In this work we will show how to predict the sequence of a large amount of clinical events by developing a temporal latent embedding model.

The chapter is organized as follows. In the next two sections we introduce the proposed models for this work. In Section 2.4 we describe details of the nephrology use case and describe the data structure in detail. In Section 8.5 we explain the experimental set ups and present its results. In Section 2.6 we explain how we successfully applied our model to an additional medical dataset consisting of breast cancer patients. Section 8.6 contains our conclusions and an outlook.

2.2 Representation Learning

Representation learning is a subfield of Machine Learning that consists of learning alternative representations of the entities that compose a dataset. These representations are usually real numbered vectors which are called “latent embeddings” or “latent representations”. A typical use case for representation learning are movie recommender systems. Using representation learning techniques, we can use a database containing movie ratings in order to learn a latent representation for each movie and for each user. Then we can, for example, take the latent representation of a movie and find its nearest neighbors in order

to identify similar movies. Moreover, some of the dimensions of the learned representations can have interpretable meanings, representing features as comedy, terror, etc. Thus, this latent representations can also be used to display interesting visualizations that wouldn't be possible with the raw data.

Besides, by computing low rank latent representations, one can obtain a sort of compressed version of the original data, meaning that the model learns to generalize over the raw input data. This feature is of course essential in any Machine Learning model and leads to improved predictions. For example, another field where representation learning has been widely used is Language Modelling. The task to be solved in Language Modelling is that of predicting the next word of a text given the previous ones. Historically, this has been accomplished by training probabilistic models that learn how likely is to observe each tuple or triplet in the dataset. However, after training such type of model, it is not possible to predict the probability of observing a tuple or triplet that is not present in the training dataset. On the other hand, if we used a model based on representation learning, it would learn a generalized latent representation for each word in the text, and therefore it would be able to make predictions about combinations of words that are not present in the training dataset.

Representation Learning can be also a good fit for modelling medical datasets, since as it happens with movies and words, there are for example multiple medications and patients that share certain features.

2.3 Temporal Latent Embeddings for Predicting Clinical Events

In this section we extend latent embedding models to be applicable to clinical data which consist of temporal sequences of high-dimensional sparse events. In particular, in our approach the latent embeddings describe the state of the patient at a given time. Another extension is that we complement the short-term memory of language models with a long-term memory by including a representation of the complete clinical history of the patient.

2.3.1 The Basic Data Structures

A recorded event in our data is based on the schema *event(Time, Patient, EventType, Value)*. *Time* stands for the time of the event and is represented as the day of the event.

Note that several events can happen at the same time. *Patient* stands for the patient ID and *EventType* for the type of the event, such as a specific diagnosis, a specific prescribed medication, a specific laboratory result and so on. For events like prescribed medications the value is equal to 1 if this particular event happens for the patient at time *Time=t* and is equal to 0 otherwise. For laboratory results such as Calcium or Blood count, we used a binary encoding and represented each measurement as three event types, i.e., *LabValueHigh*, *LabValueNormal* and *LabValueLow*.

These events can be stored in a three-way tensor $\underline{X}$ with dimensions *Time*, *Patient*, and *EventType*. The tensor entry $x_{t,i,j}$ with $t = 0,\ldots,T$, $i = 0,\ldots,I$, $j = 0,\ldots,J$ is the value of the tuple *event(Time=t, Patient=i, EventType=j, Value)*. The tensor is extremely sparse and is stored in form of a sparse data structure. The task of the learning approach is to predict tensor entries for patients in the test set. In particular we predict entries in a second tensor $\underline{\Theta}$, with the same dimensions as $\underline{X}$, that contains the patients in the test set. The relationship between both is defined by the sigmoid function $P(x_{t,i,j} = 1) = \mathrm{sig}(\theta_{t,i,j})$.

There are a number of interesting challenges in the dataset. Time plays an essential role and we are dealing with sequences of events but *absolute* time is of little value and a patient-specific normalization of time is non-trivial. Also the tensor $\underline{X}$ initially contains only data about the patients in the training dataset; our real goal of course is to obtain valid predictions for test patients which are not part of the training data without an expensive retraining of the model.

In the next subsections we will describe the Temporal Latent Embedding models we have used in the experiments. In the next subsection we describe the model which is based on the complete patient history up to time t. Subsection 8.4 then describes a Markov model that is based only on a recent history and Subsection 2.3.4 describes a combination of both.

2.3.2 Patient History Embedding

We define an aggregation tensor $\underline{\tilde{X}}$ with entries $\tilde{x}_{t,i,j}$. Here, $\tilde{x}_{t,i,j}$ is an aggregation of $\{x_{t',i,j}\}_{t'=1,\ldots,t}$, i.e., of all events that happened to patient i up to time t. In the experiments we used different aggregation functions (see Section 8.5). $\tilde{x}_{t,i,j}$ is supplemented with dimensions encoding background patient information such as age, gender and so on.

We then model

$$\theta_{t,i,j} = f_j(h^{\mathrm{hist}}_{t-1,i}).$$

Here, $h_{t,i}^{\text{hist}}$ is an r-dimensional real vector that represents the embedding of patient i at time t, based on all information observed for that patient until time t. We call r the rank of the embedding.

Since we want to apply the learned model easily to new patients, we assume that the embeddings can be calculated as a linear function of the events that are associated with patient i up to time t, with

$$h_{t,i}^{\text{hist}} = A\tilde{x}_{t,i,:}$$

where $\tilde{x}_{t,i,:}$ is a J-dimensional vector and $A \in R^{r \times J}$ is a matrix of learned weights. Thus $h_{t,i}^{\text{hist}}$ is a latent representation of the history of the patient i until time t. In a related but slightly different interpretation, we can also think of the j-th column of A as the latent embedding representation of event type j. As in other embedding approaches, the model has the ability to form similar latent representations for event types which have a similar semantics, i.e. for medications with comparable effects.

Note, that if $f_j(\cdot)$ is a linear map, we obtain a factorization approach, as used in collaborative filtering. In our experiments, the functions $f_j(\cdot)$ are nonlinear maps and are modelled by a multi-layer Perceptron (MLP) with J outputs, as also used in [8] and [65].

2.3.3 Markov Embeddings

In a K-th order Markov model, the events in the last K time steps are used to predict the event in the next time step. Markov models are used in language models where an event would correspond to an observed word [8]. Some of the leading approaches in computational linguistics [27, 61] are then using learned word embeddings to realize a number of applications and we will also pursue this approach in this chapter.

More precisely, our model is

$$\theta_{t,i,j} = f_j(h_{t-1,i}^{\text{Mar}}, h_{t-2,i}^{\text{Mar}}, \ldots, h_{t-K,i}^{\text{Mar}}).$$

Note that in this model $h_{i,t}^{\text{Mar}}$ is an an r-dimensional embedding of all the observed events for patient i at time t. Note also that, in contrast to the situation in language models, several events can happen at the same time.

As before, we assume that there is a linear map of the form

$$h_{t,i}^{\text{Mar}} = Bx_{t,i,:}$$

where $x_{t,i,:}$ is a J-dimensional vector that contains all observed events for patient i at time t.

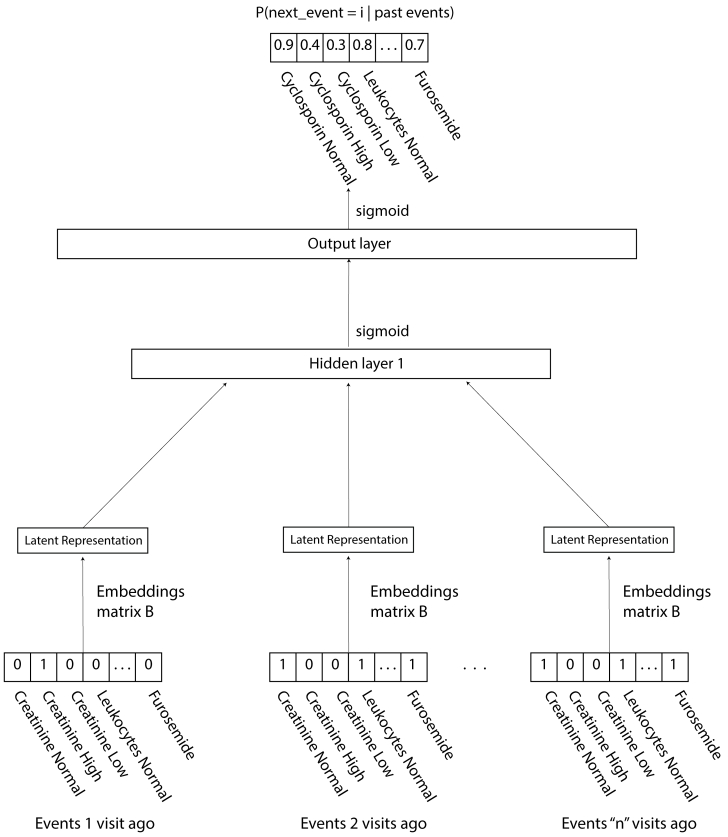

Figure 2.1: Markov embedding model for predicting sequences of clinical events by taking the previous time steps as inputs.

We can think of $h_{t,i}^{\text{Mar}}$ as the latent representation of patient i at time t based on all events that happened to the patient at time t. In contrast, $h_{t,i}^{\text{hist}}$ was the presentation of all events that happened to patient i until time t.

Again the $j-th$ column of B is representing latent embedding of event type j. The overall architecture in shown in Figure 2.1.

2.3.4 Personalized Markov Embeddings

The Markov model so far is independent of the individual patient history but it makes sense to assume that this history would be relevant for predicting events. Thus, we include $h_{t,i}^{\text{hist}}$ in the Markov model in the form

$$\theta_{t,i,j} = f_j(h_{t,i}^{\text{hist}}, h_{i,t-1}^{\text{Mar}}, h_{i,t-2}^{\text{Mar}}, \ldots, h_{i,t-K}^{\text{Mar}}).$$

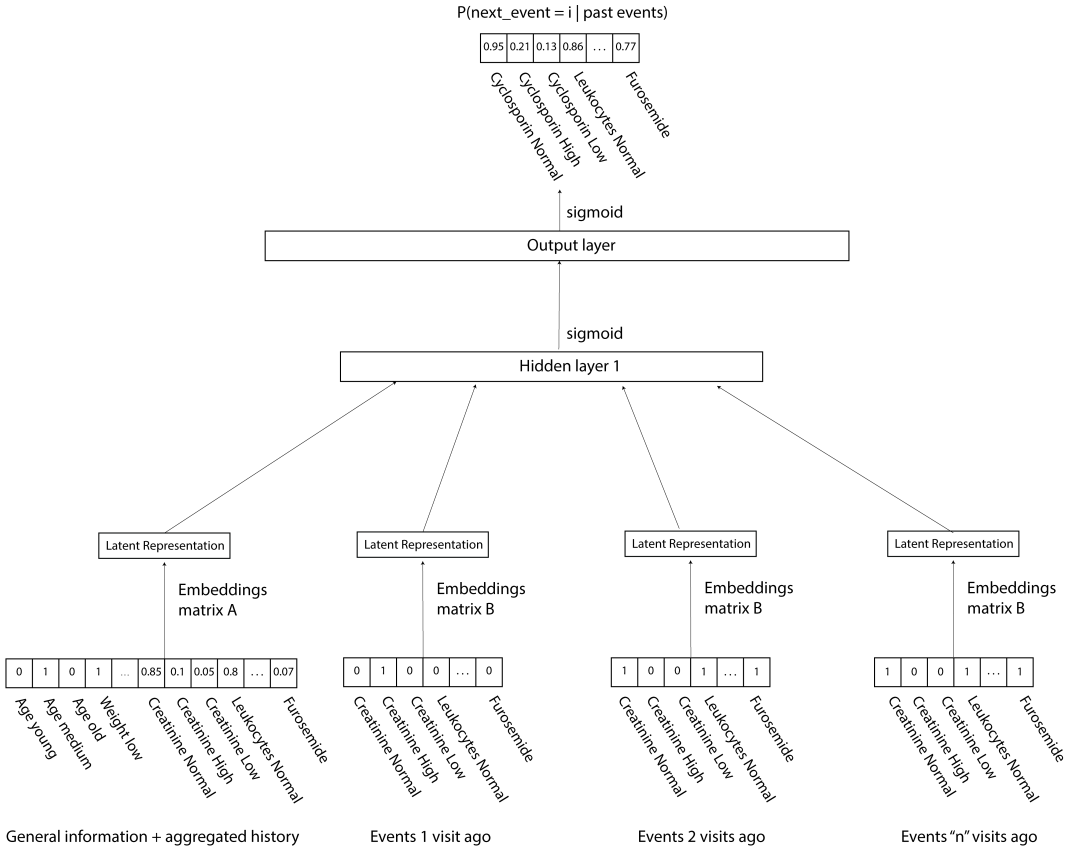

Figure 2.2: Personalized Markov embedding model. It predicts the observed events within the next time step given the patient history and the previous time steps as inputs.

The overall architecture is shown in Figure 2.2.

2.3.5 Modelling the Function

In the language models of [8], $f_j(\cdot)$ was modelled as a standard multi-layer Perceptron neural network (MLP) with one hidden layer. A similar representation was used in modelling knowledge graphs as described in [65]. We use the same MLP structure here, where we also experimented with different numbers of hidden layers. In the following, the set of all MLP parameters is denoted by $W = \{w\}$.

2.3.6 Cost Function

We derive a cost function based on the Bernoulli likelihood function, also known as Binary Cross Entropy, which has the form:

$$\text{cost}(A, B, W) =$$

$$\sum_{t,i,j \in \text{Tr}} -x_{t,i,j} \log(\text{sig}(\theta_{t,i,j})) - (1 - x_{t,i,j}) \log(1 - \text{sig}(\theta_{t,i,j}))$$

$$+\lambda_w \sum_{w \in W} w^2 + \lambda_a \sum_{l=1}^{r} \sum_{j=1}^{J} a_{i,j}^2 + \lambda_b \sum_{l=1}^{r} \sum_{j=1}^{J} b_{i,j}^2$$

Note that we added regularization terms to penalize large MLP parameters w and large embedding parameters $a_{i,j}$ and $b_{i,j}$. Here, λ_w, λ_a, and λ_b are regularization parameters. Tr stands for the training dataset and sig is the sigmoid function.

2.4 The Use Case

2.4.1 Kidney Diseases and their Treatments

Kidney diseases are causing a significant financial burden on all health systems worldwide. Here we describe the situation in Germany. It is estimated that alone the treatment of end-stage renal disease (ESRD) with chronic renal replacement therapies accounts for more than 2.5 billion Euros annually, and the incidence of dialysis-dependent renal insufficiency is rising by 5-8% each year [57]. Despite progress in diagnosis, prophylaxis and therapy of chronic kidney diseases, renal transplantation remains the therapy of choice for all patients with ESRD. Kidney transplantation leads to a significant improvement of quality of life, to substantial cost savings and most importantly to a significant survival benefit in relation to all other renal replacement therapies. Only approximately 2300 kidney transplantations were performed in Germany in 2013 but more than 8000 patients are registered on the waiting list for a kidney transplant [70]. With excellent short term success rates, nowadays the reduction of complications and the increase of long-term graft survival are the main goals after transplantation, especially on the background of the dramatic organ shortage. It is not only important to reduce - or better avoid - severe and/or life-threatening complications such as acute rejection, malignancy and severe opportunistic infections, but it is also of utmost importance to ameliorate the many other serious side

effects, which increase cardiovascular risk, decrease renal function, necessitate costly co-medication or hospitalisations and also have an impact on the quality of life after successful transplantation.

Despite the fact that renal transplantation is much cheaper than regular dialysis treatment it is a complex and costly procedure. Due to the outlined complexities, patients should remain in life-long specialized posttransplant care. Patients have not only to take immunosuppressants, but also have to take numerous drugs for prophylaxis and treatment of pre-existing and/or concomitant diseases, which are at least in part aggravated by the immunosuppressants. As a consequence most patients have to take 5-10 different medications every day during their entire life. The many drugs and the multiple side effects of the routinely administered medication are causing a substantial cost burden. There is not only a medical need but also a financial necessity to reduce side effects, diagnostic procedures, therapeutic interventions, hospitalisations and ultimately improve patient safety. This will directly lead to a better quality of life, cost savings and better allocation of medical resources.

2.4.2 Relevance of Event Modelling

The long-term goal of the research described in this chapter is to improve patient treatment by, e.g., prescribing the most effective drugs to the patient my minimizing side effects. Particularly in focus are drug-drug interaction (DDI) and adverse drug reactions (ADR) in patients after renal transplantation. Of high interest are the effects of decisions on key outcome parameters such as patient and graft survival, renal function as well as hospitalisations. Lastly, the goal is to implement a clinical decision support system directly into the electronic patient file, in order to prevent dangerous DDI, reduce dosing errors and provide the physician and patient with timely and adequate information on new prescriptions.

2.4.3 TBase®

In close collaboration with the department of Artificial Intelligence of the Humboldt University, the Charité - Universitätsmedizin Berlin developed an electronic patient record (TBase®) for renal allograft recipients in 1999. The main idea was to combine a database for the daily patient care on the one hand with a validated database for medical research of the other hand. The combination of daily medical routine with a research database was the key concept, in order to collect data of high quality, which are constantly validated by

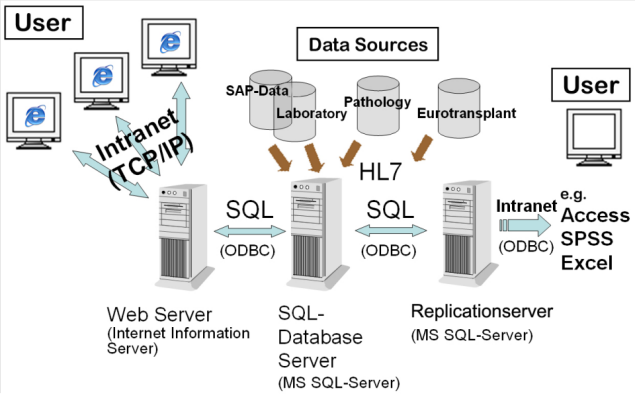

Figure 2.3: TBase®architecture

the user. Due to clinical needs only accurate and reliable data can be used in daily routine practice. By this means, we have created a continuous internal validation process and almost completely avoid missing data. Since 2000 TBase®is used in the clinical routine of the Charité and all relevant patient data is automatically transferred. Due to the increase of the options of medical diagnostics, the extent of the information of the clinical data has also increased dramatically. The elaborate and flexible structure (see Figure 2.3) of the patient record and the database made it possible to integrate a large number of electronic data of several subsystems with different data structures over the years. Currently TBase®automatically integrates essential data from the laboratory, clinical pharmacology, nuclear medicine, findings from radiology and administrative data from the SAP-system of the Charité. TBase®is now under patronage of Deutsche Transplantationsgesellschaft (DTG) and Eurotransplant, Leiden, The Netherlands, and was implemented in 8 German transplant centres. Figure 2.4 provides an impression of the schema of TBase®.

2.5 Experiments

2.5.1 Setup of the Experiments

The data contains every event that happened to each patient concerning the kidney failure and all its associated effects, including prescribed medications, hospitalizations, diagnosis, laboratory tests and so on. In this chapter we will consider events from year 2005 and

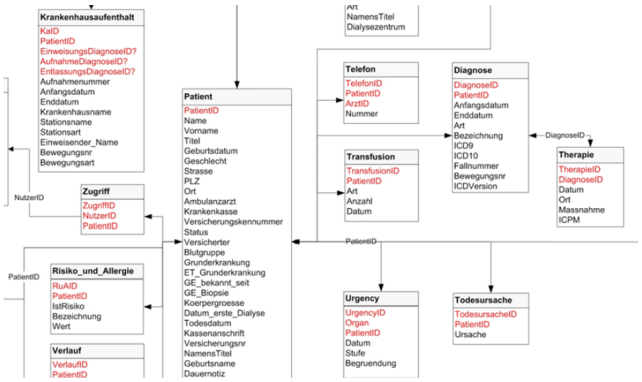

Figure 2.4: View on the TBase®Schema

onwards due to the improvement of the data quality from that year. Also, in order to have a better control of the experiments, we will work with a subset of the variables available in the dataset. Specifically, we will try to model three aspects of the patient evolution:

1. Medication prescriptions: which medications are prescribed in each situation.

2. Ordered laboratory tests: which laboratory tests are ordered in each situation.

3. Laboratory test results: which will be the outcome of the ordered laboratory tests.

Each entry in the database is labelled with the date in which the event happened. Our task will consist in predicting all the events that will happen to a patient on his or her next visit to the clinic given his past visits, as illustrated in Figure 2.5.

A very common situation is that the patient gets some laboratory tests done during the morning, and then based on the results of those tests, the doctor prescribes some medications to the patient in the afternoon. Therefore, we can define a second type of experiment by only considering days that have both laboratory tests performed and medications prescribed, and assuming that the laboratory tests always happen before the medications. Specifically, we will try to predict which will be the medications prescribed in the afternoon given the results of the laboratory tests performed in the morning and the events that happened in the previous visits. This way we can see how the model behaves in intra-day predictions. Figure 2.6 shows a representation of the experiment.

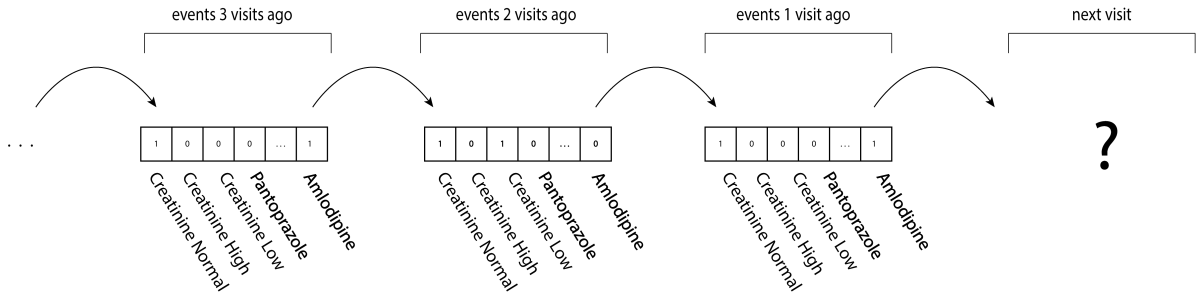

Figure 2.5: Full visit predictions. We predict all the events that will happen within the next visit given the previous visits.

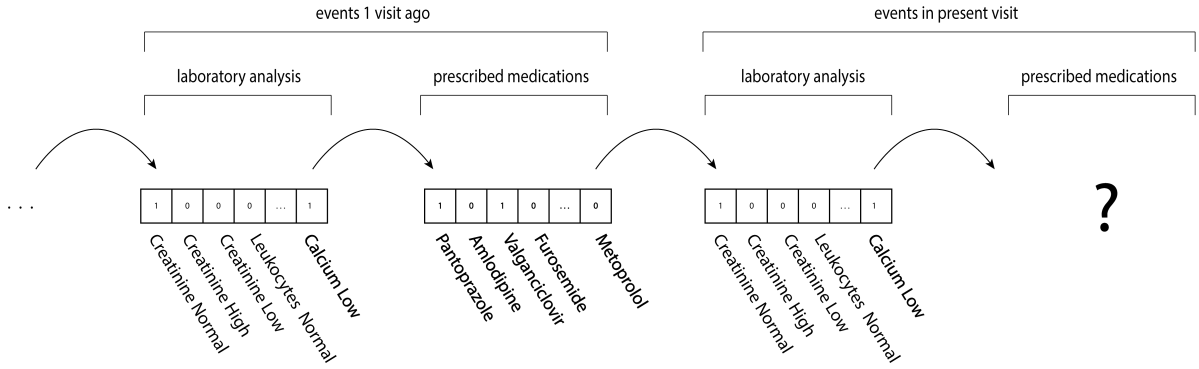

Figure 2.6: Intra-day predictions. We predict the medications that will be prescribed in the afternoon given the laboratory analysis that were performed in the morning and the previous visits.

After selecting the subset of the dataset that we will use and performing the binary encoding, our pre-processed dataset consists of a table where each row represents one visit to the clinic. Each of these rows belongs to a patient, has an associated date and contains all the events that occurred during that visit in binary format. An example of how our pre-processed data look like can be found in Figure 2.7.

2.5.2 Hyperparameter Fitting

The model contains several hyperparameters that need to be optimized, being the most relevant ones the rank r of the embeddings, the order of the Markov model K, the number of hidden units in the Neural Network, the learning rate and the regularization parameters.

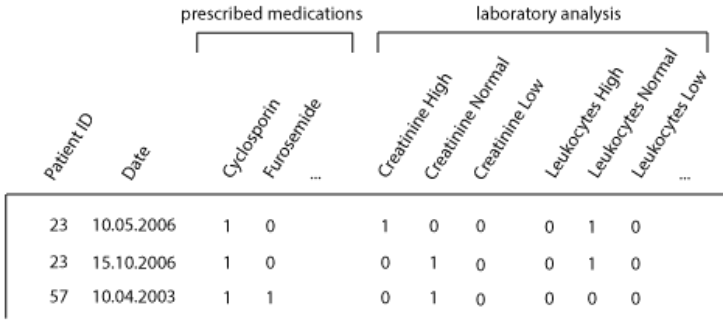

Figure 2.7: Example of pre-processed data.

In order to fit these hyperparameters, we randomly split the data into three subsets: 60% of the patients were assigned to the training set with totally about 100 thousand visits, 20% of the patients were assigned to the validation set and another 20% to the test set, with approximately 33 thousand visits each. Note that, under this configuration, we evaluate the performance of the model by predicting the future events of patients that the model has never seen before, and therefore increasing the difficulty of the task.

In Figure 2.8 we can see how the area under the Precision-Recall curve on the validation set improves as we increase the order of the Markov model K. We observe that the performance stabilizes with an input window of size six. A 6-th order Markov model (without the personalization) has around 28 thousand inputs (4666 input events multiplied by 6 time steps). The number of outputs of the Neural Network is 2383, i.e. 2383 events are predicted.

2.5.3 Baseline Models

We will compare the performance of our model with various classic Machine Learning algorithms. Specifically, our baseline models will be: Naive Bayes classifier, K-nearest neighbor classifier and Logistic Regression. Additionally, we will also use what we named "constant predictor", which consists in predicting always for each event the occurrence rate of such event (thus the most common event is given the highest probability of happening, followed by the second most common event, and so on). Random Forests were also considered to be included in this work, but after some trials they were discarded due to the excessive amount of time they required to be trained with this dataset, due to the large number of events to be predicted (nevertheless in the few experiments we performed with them, they never got to outperform our proposed models). When comparing the performance

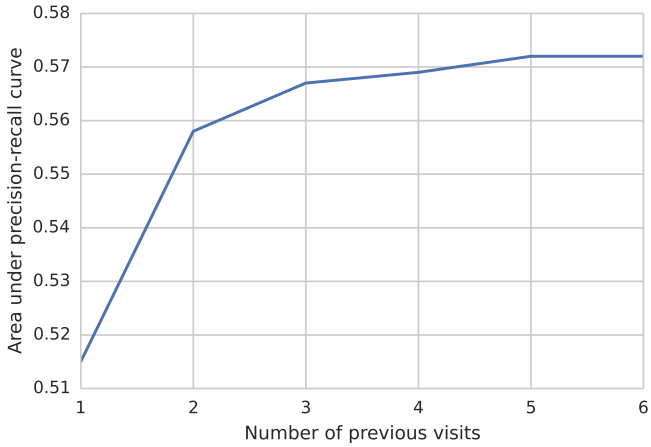

Figure 2.8: Area Under the Precision Recall Curve improves as we increase the number of past visits (order of the Markov model K) used to predict the events that will be observed in the next visit.

between these models, we report for each model the mean area under the Precision-Recall curve (AUPRC) and mean area under Receiver Operating Characteristics curve (AUROC) together with their associated standard errors after repeating each experiment ten times with different random splits of the data. We made sure that these ten random splits were identical for each model. Most of these baseline models were taken from Scikit-learn [74], which is the main open source machine learning library for the Python programming language.

2.5.4 Model Training and Evaluation

We trained the proposed models by using mini-batch Adaptive Gradient Descent (AdaGrad) [30] combined with an early stopping strategy and using a mini-batch size of 128 samples. Our main goal will be to maximize the area under the Precision-Recall curve (AUPRC) of our predictions. We chose this score due to the high sparsity of the data (the density of ones is around 1%) and because we are mainly interested in predicting the very small amount of events that will happen, as opposed to the task of predicting which events will not be observed. Nevertheless, we will also report the area under Receiver Operating Characteristics curve (AUROC) because it is often reported in related scenarios.

The proposed models were implemented in Theano [12, 5], which is a graph-based

computation library, especially well suited for training Neural Networks. The experiments were conducted using a Intel(R) Xeon(R) CPU E7-4850 v2 processor with 1TB of RAM and 48 cores at 1.2 Ghz with 2 threads per core. The reported computation times were all achieved using one thread.

2.5.5 Full Visit Predictions

As explained earlier in this section, our first experimental setting consists in predicting all the events that will happen to the patients during their next visit to the clinic given the events that were observed in their previous visits, as it is illustrated in Figure 2.5.

Therefore, we predict the events that will happen to a patient in her or his next visit to the clinic given the events that were observed in her or his six previous visits to the clinic, i.e. $K = 6$. Table 8.1 shows the results obtained after repeating the experiments with ten different random splits of the data. We can see that the Markov embedding model, which corresponds to the architecture shown in Figure 2.1, outperforms all our baseline models. Our proposed Markov embedding model obtained an AUPRC score of 0.574, being Logistic Regression the second best model with an AUPRC score of 0.554. We can also see how the random predictor achieved a very low AUPRC score due to the high sparsity of the data, which means that optimizing the AUPRC for this dataset is a hard task.

In the last column of Table 8.1 we also report the time that it took to train for each model with the best set of hyperparameters in the first random split. Note that one of the advantages of the proposed model is that the rank of the embeddings matrix $\boldsymbol{B}$ can always be reduced in order to decrease the computational cost required to train the model. Besides, given constant hyperparameters, the parameters of the model will increase linearly with the amount of different event types present in our dataset (e.g. number of medications, number of diseases...), whereas the parameters of other models such us the Logistic Regression will grow quadratically in this situation since for every additional event that we include we are adding both one input and one output.

We repeated the same experiment with the Personalized Markov Embedding model as represented in Figure 2.2. The additional information that we input to the model is composed of the aggregated history and general information of each patient. In order to create the aggregated history, for each sample that we input to the model we create a vector composed of the sum of all the events that are recorded for that particular patient until the date of the visit we want to predict. Our experiments showed that instead of directly using this count of the data as long term memory, we have two options that work better. The first

Table 2.1: Scores for full visit predictions. AUPRC stands for Area Under Precision-Recall Curve. AUROC stands for Area Under ROC Curve.

	AUPRC	AUROC	Time (hours)
Markov Embeddings	0.574 ± 0.0014	0.977 ± 0.0001	6.11
Logistic Regression	0.554 ± 0.0020	0.970 ± 0.0005	4.31
KNN	0.482 ± 0.0012	0.951 ± 0.0002	17.74
Naive Bayes	0.432 ± 0.0019	0.843 ± 0.0015	39.1
Constant predictions	0.350 ± 0.0011	0.964 ± 0.0001	0.001
Random	0.011 ± 0.0001	0.5	-

option consists in computing the frequency of appearance of each event by dividing each row of the memory by the number of visits used to make the count. The second option consists in normalizing the count between 0 and 1. We will use both the appearance frequency of each event and the normalized count as our long term memory. Regarding the background information, it is composed of static or slow changing variables that we also converted to a binary format. Specifically, the background information is composed of the following variables: age, gender, blood type, time from first dialysis, time from the first time the patient was seen, weight and primary disease. We can see in Table 8.2 how the personalization of the Markov embedding model improved its performance. During our experiments, we observed that among all the variables that compose the additional information used in this experiment, the inclusion of the frequency of appearance of each event is the factor that contributed most to the improvement of the performance of the model. Last row in 8.2 shows the performance of the model when making the predictions using just the aggregated patient history as input, as described in Section 2.3.2.

Regarding the architecture of the personalized Markov embedding model, we also tested the option of having just one embeddings matrix shared between the long term memory and the visits within the time window, i.e. $A = B$, but we found that the best strategy for our use case is to use separate embeddings matrix for the long term memory and the background information as it is shown in Figure 2.2.

We also tried to initialize the embedding matrices by using an autoencoder. This brought a speed up of around 30% to the optimization process of the model. However, this advantage vanished when we considered both the model optimization time and the

Table 2.2: Scores for full visit predictions with and without long term memory and background information. AUPRC stands for Area Under Precision-Recall Curve. AUROC stands for Area Under ROC Curve.

	AUPRC	AUROC
Personalized Markov embeddings	0.584 ± 0.0011	0.978 ± 0.0001
Markov embeddings	0.574 ± 0.0014	0.977 ± 0.0001
Patient history embedding	0.487 ± 0.0016	0.974 ± 0.0002

training time of the autoencoder.

2.5.6 Intra-day Predictions

Our second experiment type consists in predicting which medications will be prescribed in the afternoon given the results of the laboratory tests performed in the morning and the events that happened in the six previous visits. Figure 2.6 shows a representation of the experiment. The architecture of the model will be similar to the one for the Markov embedding model (Figure 2.1), but including one more time step in the input window that will contain the information regarding all the observed events in the present day. Therefore, for this experiment the order of the Markov model K will be equal to seven, instead of six as it was in the case of full visit predictions. We can see the result of the experiment in Table 5.3, which shows that also in this setting our proposed model outperforms the baseline models. The Markov embedding model for intra-day predictions achieved an AUPRC score of 0.277, which is lower than the score achieved when doing full visit predictions because the dataset is even more sparse when we only take into account the medications. Logistic Regression is again the second best result, and we can also observe how in this case the performance of the constant predictor is almost as bad as the random predictor, which means that this is even a harder task than the full visit predictions.

Another interesting experiment is to compare this result with the one obtained when doing full visit predictions. That is, we will measure the performance of predicting medication prescriptions both considering the laboratory tests performed in the same day and not considering them. Table 5.4 shows that incorporating intra-day information actually improves the performance of the predictions.

Table 2.3: Scores for intra-day predictions. AUPRC stands for Area Under Precision-Recall Curve. AUROC stands for Area Under ROC Curve.

	AUPRC	AUROC
Markov embeddings intra-day	0.277 ± 0.0026	0.935 ± 0.0007
Logistic Regression intra-day	0.238 ± 0.0041	0.916 ± 0.0014
KNN	0.184 ± 0.0027	0.873 ± 0.0002
Naive Bayes intra-day	0.231 ± 0.0013	0.686 ± 0.0020
Constant predictions intra-day	0.008 ± 0.0013	0.564 ± 0.0064
Random intra-day	0.006 ± 0.0064	0.5

Table 2.4: Scores for intra-day predictions with and without considering the present day. AUPRC stands for Area Under Precision-Recall Curve. AUROC stands for Area Under ROC Curve.

	AUPRC	AUROC
Markov embeddings intra-day	0.277 ± 0.0026	0.935 ± 0.0007
Markov embeddings	0.250 ± 0.0022	0.931 ± 0.0006

Besides, as we did with full visit predictions, we will make intra-day predictions incorporating a long term memory and background information of the patients. Table 2.5 shows how we improved the performance of the predictions with the personalized Markov embedding model.

2.5.7 Sensitivity Analysis

We performed a sensitivity analysis in order to evaluate how the model reacts to changes in the inputs. We performed this analysis using the medication named Tacrolimus, because it is one of the main immunosuppressants used in our database but it is not as frequent as other immunosuppressants such as Cyclosporin.

When doing the intra-day predictions as illustrated in Figure 2.6, and if we look exclusively at the score obtained in the prediction of Tacrolimus prescription (i.e. predicting whether or not Tacrolimus prescription will be observed next), we obtain an AUPRC score

Table 2.5: Scores for intra-day predictions with and without memory and background information. AUPRC stands for Area Under Precision-Recall Curve. AUROC stands for Area Under ROC Curve.

	AUPRC	AUROC
Personalized Markov embeddings intra-day	0.289 ± 0.0027	0.938 ± 0.0005
Markov embeddings intra-day	0.277 ± 0.0026	0.935 ± 0.0007

of 0.629, whereas the random prediction score 0.160. The sensitivity analysis will consist in suppressing one by one the events in the input and check how the absence of such input affects to the AUPRC score.

After performing this analysis we rank our input variables according to how much the AUPRC score of predicting Tacrolimus prescription was degraded when suppressing each of them. Even though this is a simplified analysis since we do not analyze how each variable influences the output when combined with other variables, we can infer that the higher a variable is ranked, the higher is the importance that has been assigned to it by our model for this task.

Most of the prescriptions of Tacrolimus present in our database correspond to an increase or decrease of the amount of medication that a patient is taking. The dosage of Tacrolimus that a patient takes has to be adjusted when certain criteria are met. The factors that the physicians take into account to decide whether or not the dosage of Tacrolimus has to be changed are the amount of Tacrolimus in blood and the excess of Creatinine in blood. Out of almost 5000 events, the laboratory results for "Low Tacrolimus", "High Tacrolimus" and "Normal Tacrolimus" occupy the positions second, third and fourth respectively in our sensitivity ranking. The laboratory result of "High Creatinine" occupies the position number 27. Therefore we can see how the model has learnt to predict the prescription of Tacrolimus giving a very high importance to the same observations that the physicians use. Moreover, other factors that are also correlated with the prescription of Tacrolimus are also present in the top 10 entries of the sensitivity ranking. For example in the position number 8 we find "High C-reactive protein", which is an infection marker that, when observed, indicates that the Tacrolimus dosage has to be reduced. Also in position 10 we find "High Glucose" which is a side effect of Tacrolimus that often leads to

patient ID	date	Cluster 1	Cluster 2	Cluster 3	...
1	05/06/2014	1	1	0	
1	20/08/2014	0	1	0	
1	13/11/2014	0	0	0	
2	12/06/2014	1	0	0	
...	...	...	...	...	

Figure 2.9: Data sample.

the reduction of the Tacrolimus dosage.

2.6 Breast Cancer Dataset

2.6.1 The Dataset

As we mentioned already, one of the main issues when working with medical data, is that there is a huge amount of information that is stored in form of unstructured data. Vogt et al. [99] developed an algorithm that allows to generate sentence clusters out of a set of clinical notes composed of free text. Thus, this algorithm converts the unstructured data if the clinical notes into structured data. As a result of the process, we will have for each patient a binary matrix indicating which clusters has been observed on each visit. Figure 2.9 shows an example of how the final dataset looks like.

For our work, in collaboration with the Memorial Sloan Kettering Cancer Center of New York, we took a set of breast cancer patients, composed of 11000 patients approximately. The total number of medical notes was 180000. After the clustering process, a total of 679 clusters were found.

2.6.2 Experimental setting

The experiment will consist of predicting which sentence clusters will be observed on the next visit to the clinic, given the previously observed cluster.

We randomly split the data into three sub sets: 60% for training, 20% for validation and 20% for test. Adagrad optimization method was used to fit the Markov Embeddings Model. Cross validation was used for setting the rest of the hyperparameters such as number of units in the hidden layer, rank of the latent embeddings, learning rate, etc.

2.6.3 Results

Table 2.6 shows the results of the experiment. KNN stands for K Nearest Neighbors. We can see how the Markov Embeddings Model outperformed the other models, although its AUPRC score is very close to the one obtained by the Logistic Regression.

Table 2.6: Scores for the task of predicting which sentence clusters will be observed on the next visit to the clinic.

	AUROC	AUPRC
Markov embeddings	0.952	0.533
Logistic Regression	0.929	0.532
KNN	0.950	0.484
Random	0.5	0.0049

It must be noted that in this experiment we did not used static information of patients, but just the dynamic information, composed of the sequence clusters that has been previously observed. Due to the nature of the scenario, the hospital is also storing genetics information concerning these patients, and therefore it would be possible to integrate such data into the model.

2.7 Potential Extensions

We will try to improve the model by introducing other elements that proved to be successful in deep Neural Networks such as drop out regularization and temporal convolutional layers. We will also explore the possibility of including additional information in the model such as the size of the time gap between the visits.

Besides, [61] showed that Recurrent Neural Networks provide the best performance in the task of language modelling. Therefore we will explore such models for our use case.

Regarding the data, we will extend our model to predict more event types within this dataset, and we will also apply our model to other datasets and use cases.

Our project also serves to encourage the TBase system to collect more information that would be valuable for decision support, such as patient symptoms and a precise time stamp for each event. Future work will also include the incorporation of textual information as

present in pathology reports and information from molecular tests, e.g., genetics. Finally, we plan to make more extensive use of background ontologies which for example can be used to map different medications with identical active components to a common representation.

2.8 Chapter summary

In this chapter we presented a model capable of predicting clinical events that is scalable and provides an acceptable performance for our current use case, which consist of modelling a subset of the variables that compose the evolution of the patients in our dataset.

Our work already lead to new requirements for improving the medical documentation. For example a detailed documentation of the patients symptoms would be a very valuable information for improving the model.

We showed how the proposed model performed better than our baseline models both making full visit predictions and intra-day predictions. We also showed how to integrate both the background information of each patient and a long term memory in order to improve the performance of the model.

The model presented currently predicts common practice in a clinic which can already be useful in many ways, for example in alerting staff in case of unusual decisions. Of course the ultimate goal of a clinical decision support system should be not just replicating the decisions that are most often taken by the physicians in each situation, but to provide recommendations that lead to the best outcome possible. The basis for achieving this goal is a predictive model as presented in this chapter.

Chapter 3

Review of Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of Neural Network specifically designed to model sequential, and therefore we will make use of it in the next chapters of this thesis. In RNNs, the hidden state of one time step is computed by combining the current input with the hidden state of the previous time step. This way they can learn to remember events from the past that are relevant to predict future outcomes. In Figure 3.1 we can see the architecture of an RNN, whereas Figure 3.2 shows the unfolded representation of an RNN.

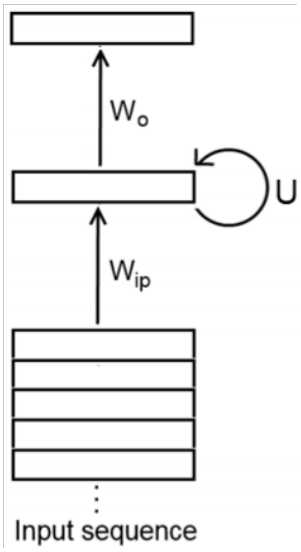

Figure 3.1: Recurrent Neural Network.

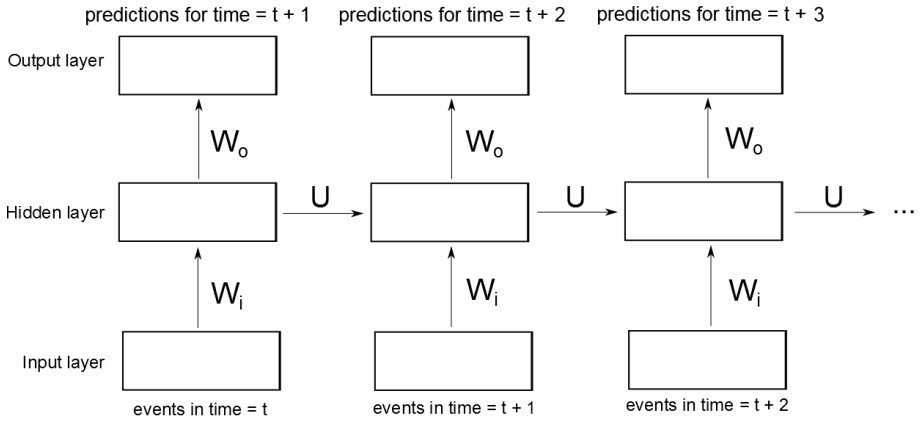

Figure 3.2: Unfolded representation of a Recurrent Neural Network.

As opposed to the models based on Feedforward Neural Networks, when sequential data is modelled using RNNs, we do not need to specify how many time steps from the past we want to consider to predict future events. Therefore, RNNs can theoretically learn to remember any event (or combination of events) that occur in the life of the patient that is useful to predict future events. This feature can be very valuable when our clinical data set presents such kind of long term dependencies.

Another advantage of using RNNs for learning with sequences is that, given a new patient, we can start predicting future events for such patient right after his or her first visit to the clinic. On the other hand, if we model our data with a Feedforward Neural Network and we have decided to use the "n" previous visits to predict future events, we will have to wait until we have accumulated at least "n" visits for a patient to start predicting his or her future events. In some scenarios, it can be very useful to have a system with the ability of making predictions with sequences of different length.

More formally, the output of an RNN is computed as:

$$\hat{y}_t = \sigma\left(W_o h_t\right) \tag{3.1}$$

where σ is some differentiable function, usually the logistic sigmoid function or the hyperbolic tangent, and h is the hidden state of the Neural Network.

Given a sequence of vectors $x = (x_1, x_2, \cdots, x_T)$, the hidden state of an RNN is computed the following way:

$$h_t = f\left(h_{t-1}, x_t\right) \tag{3.2}$$

We will summarize the most common options for the f function in the next sections.

3.1 Standard Recurrent Neural Network

This is the classic way of updating the hidden state in RNNs:

$$h_t = \sigma\left(W x_t + U h_{t-1}\right), \tag{3.3}$$

In order to compute the value of the hidden state at time t, we perform a linear combination of the hidden state of the previous time step h_{t-1} with the current input to the network x_t. This way the Neural Network can learn to remember specific events observed in the past by encoding them in the hidden state, in order to use such information to improve the predictions in the present time step.

3.2 Vanishing Gradient Problem

As we mentioned earlier, one of our main interests for using RNNs is to capture long-term dependencies. The ability to remember events that occurred a long time ago can be specially useful for some medical applications where events observed at any point in the life of the patient can be very informative about future events that will be observed.

However, it was observed by Bengio et al. [9] that it is not possible for standard RNNs to capture such long-term dependencies due to the vanishing gradient problem. The vanishing gradient problem shows that, as we propagate the error through the network to earlier time steps as represented in 3.3, the gradient of such error with respect to the weights of the network will tend to 0.

One of the reasons that cause this issue is that, for each additional time step that we go into the past, we multiply the gradient of the error by the derivative of the sigmoid. This derivative has a maximum value of 0.25, as seen in Figure 3.4. Other non linear functions often used in NNs such as the hyperbolic tangent also present similar small gradients. Thus, as after a few time steps, we have multiplied the gradient multiple times by very small numbers and therefore it tends to 0. Pascanu et al. recently published a thorough study on the subject [72]. Besides, in RNNs, the hidden state is repeatedly multiplied by the same matrix on each time step. This recurrent behavior often makes RNNs unstable, and the gradients tend to either explode or vanish [10].

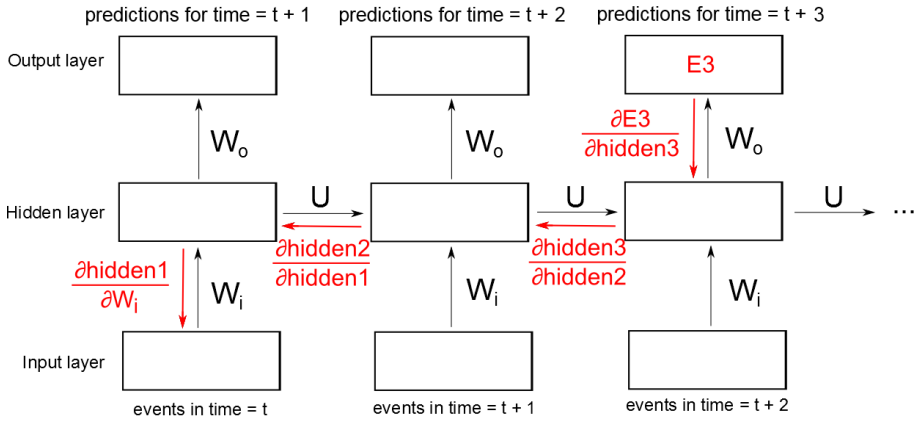

Figure 3.3: Simplified representation of backpropagation through a Recurrent Neural Network. "hiddenX" represents the internal state of the network on the Xth temporal state.

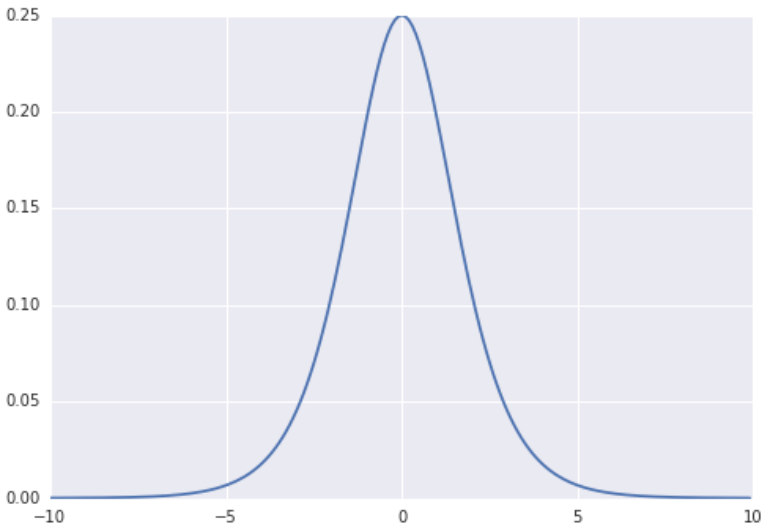

Figure 3.4: Sigmoid derivative.

3.3 Long Short-Term Memory units

In order to alleviate the gradient vanishing problem, Hochreiter *et al.* [41] developed a gating mechanism that dictates when and how the hidden state of an RNN has to be updated.

There are different versions with minor modifications regarding the gating mechanism

in the long short-term memory units (LSTM) units. We will use in here the ones defined by Graves *et al.*[39].

$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \tag{3.4}$$
$$r_t = \sigma\left(W_{xr}x_t + W_{hr}h_{t-1} + W_{cr}c_{t-1} + b_r\right) \tag{3.5}$$
$$c_t = r_t \odot c_{t-1} + i_t \odot \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \tag{3.6}$$
$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o\right) \tag{3.7}$$
$$h_t = o_t \odot \tanh(c_t) \tag{3.8}$$

where $\odot$ is the element-wise product and i, r and o are the input, forget and output gates respectively. As it can be seen, the gating mechanism regulates how the current input and the previous hidden state must be combined to generate the new hidden state.

The main differences between this implementation of the LSTM units and the first one published by Hochreiter *et al.*, is that the original implementation didn't include forget gates and the activation function was the sigmoid function instead of tanh.

LSTM units have been successfully used many times before [21, 103]. Indeed they have become the de facto standard for RNN implementations. The only downside of LSTM units with respect standard RNN updates is the increase of network parameters, which makes them more computationally expensive. However, the improvement they bring in the quality of the predictions makes them the preferred choice in most applications.

3.4 Gated Recurrent Units

Another gating mechanism named Gated Recurrent Units (GRUs) was introduced by Cho et al. [22] with the goal of making each recurrent unit to adaptively capture dependencies of different time scales. We follow the equations as defined by Chung et al. [25]:

$$r_t = \sigma\left(W_r x_t + U_r h_{t-1}\right) \tag{3.9}$$
$$\tilde{h}_t = \tanh\left(W x_t + U\left(r_t \odot h_{t-1}\right)\right) \tag{3.10}$$
$$z_t = \sigma\left(W_z x_t + U_z h_{t-1}\right) \tag{3.11}$$
$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t \tag{3.12}$$

where z and r are the update and reset gates respectively.

As it can be seen, GRUs present an architecture less complex than LSTM units, and the former usually perform at least as good than the latter.

In [25], Chung et al. compare the performance provided by standard RNNs, LSTM units and GRU units, using multiple datasets. In our experiments, GRU units outperform the other models most of the times.

Chapter 4

Predicting Clinical Events by Combining Static and Dynamic Information Using Recurrent Neural Networks

4.1 Exploiting Long Term Temporal Dependencies in Medical Data

In chapter 2, we used an approach based in Feedforward Neural Networks which did not exploit the sequential nature of clinical data. Therefore, in this chapter, we will explore alternatives in order to model the long term temporal correlations that may exist on the data together with its static features.

Particularly, there is a notable parallelism among the prediction of clinical events and the field of Language Modelling, where Deep Learning, a class of Neural Networks with multiple hidden layers, has also proven to be very successful. One could imagine that each word of a text represents an event. Therefore a text would be a stream of events and the task of Language Modelling would be to predict the next event in the stream. For this reason, we can get inspired by Language Modelling to create models that predict clinical events. However, the medical domain has a set of characteristics that make it an almost unique scenario: multiple events can occur at the same time, there are multiple sequences (i.e. multiple patients), each sequence has an associated set of static variables and both

inputs and outputs can be a combination of Boolean variables and real numbers. For these reasons we need to develop approaches specifically designed for the medical use case.

For this work we use the same dataset that was used in chapter 2. As explained, it is a large dataset collected from patients that suffered from kidney failure. In this chapter we will make use of a piece of information that we didn't consider before: endpoints. There are a set of endpoints that can occur after a transplantation and it would be very valuable for the physicians to be able to know beforehand when one of these is going to happen. Specifically we will predict whether the patient will die, the transplant will be rejected, or the transplant will be lost. For each visit that a patient makes to the clinic, we will anticipate which of those three events (if any) will occur both within 6 months and 12 months after the visit.

In order to accomplish the prediction of these endpoints, we developed a new model based on Recurrent Neural Networks (RNNs). The main advantage of our model is its ability to combine static and dynamic information from patients. This capability is very important for medical applications, since most of the clinical datasets present some background information about the patients (e.g. gender, blood type, main disease, etc.) combined with dynamic information that is recorded during the multiple visits to the clinic (e.g. results of the laboratory tests, prescribed medications, etc.).

The chapter is organized as follows. In the next section we discuss alternative approaches for endpoint predictions. One example is a Feedforward Neural Network that is also able to deal with dynamic and static information. This model will be used as our main benchmark. In Section 4.2 we describe details of the nephrology use case and explain why anticipating these endpoints could be of great value for physicians. Section 4.3 introduces the proposed models for this work, starting with a brief overview on RNNs followed by the specific details of our work. In Section 4.4 we explain the experimental set ups and present our results. Section 4.5 contains our conclusions and an outlook.

In 2 we introduced the Temporal Latent Embeddings model which is based on a Feedforward Neural Network. This model outperforms its baselines for the task of predicting which events will be observed next given the previous events recorded (i.e. the goal was to predict which laboratory analyses and medication prescriptions will be observed in the next visit for each patient).

Overall, the Temporal Latent Embeddings resembles an n-th order Markov model, and due to its architecture, it requires to explicitly define the number of time steps in the past that we want to consider in order to predict the next step. In some scenarios,

this constraint can actually be an advantage since many recent papers have shown how attention mechanisms do actually improve the performance of the Neural Networks [21, 103]. The Temporal Latent Embeddings model puts all its attention on the last n samples and therefore it provides an advantage over RNNs for datasets where the events that we want to predict are dependent just on the n previous events. However, in order to capture long term dependencies on the data with this model, we have to aggregate the whole history of each patient in one vector (e.g. computing the mean values of each laboratory measurement), and therefore many long-term dependencies can be lost in this aggregation step (e.g. a very high value in one measurement followed by a very low value).

In recent work, Choi et al. [24] used an RNN with Gated Recurrent Units (GRUs) combined with skip-gram vectors [60] in order to predict diagnosis, medication prescription and procedures. However in this work they follow a standard RNN architecture that takes sequential information as input, whereas the static information of the patients was not integrated into the Neural Network. However, due to its nature, medical data will always contain both static and dynamic features, and therefore it is fundamental to develop algorithms that can combine and exploit both types of data.

Outside of the medical domain, there are multiple examples where RNNs have been successfully applied to sequential datasets in order to predict future events. For example, in Natural Language Processing RNNs are commonly used to predict the next word in a text or even full sentences [61, 93]. However to the best of our knowledge none of them includes both sequential and static information, which is the typical setting in sequences of clinical data.

4.2 Kidney Transplantation Endpoints

Chronic kidney disease is a worldwide health burden with increasing prevalence and incidence. It affects multiple organ systems and may result in end-stage renal disease. The growing number of patients with end-stage renal disease requiring dialysis or transplantation is a major challenge for health-care systems around the world [28]. For most patients, kidney transplantation is the treatment of choice offering lowest morbidity, significant survival benefit, lowest costs and highest quality of life. The main goals after transplantation are the reduction of complications and the increase of long-term graft and patient survival. However, kidney transplant recipients are at high risk of severe complications like rejections, infections, drug toxicity, malignancies and cardiovascular diseases. The majority of

patients have to take 5-10 different medications every day during their entire life. Due to complexities of post-transplant management, kidney transplant recipients should remain in life-long specialized care. The medical records of kidney transplant recipients mostly cover a very long history of disease (years to decades) and include a vast number of diagnoses, symptoms, results, medications and laboratory values.

Due to this complexity of medical data, decision making is complex and difficult in the clinical practice of kidney transplantation. Considering the load of treating 20-40 patients per day, it is generally not possible for the medical practitioner to review all available medical information during every bed side visit or outpatient consultation. Early prediction of clinical events on the base of current data, as proposed in our proposed solution, can lead to informed decision making on how to best choose future therapy and identify current problems. Thus our proposed solution can help to avoid complications and improve survival of the patient and graft, reduce morbidity and improve health related quality of life.

Assessing the risk of graft failure or death in renal transplant recipients can be crucial for the individual patient management by detecting patients that need intensified medical care. Integrating a computerized tool to predict relevant end points (deteriorating of the graft function, infections, rejections, graft loss, death) on the base of all available medical data may be of great benefit in the clinical routine and provide medical professionals with significant decision support.

Detecting a higher risk of graft failure or death 6-12 months in advance may timely allow identifying toxicities, side effects, interactions of medications, infections, relevant comorbidities and other complications. Early detection permits timely intervention with a chance of improved outcome. Furthermore, predicting future rejections during consultations enables the medical practitioner to change immunosuppressive medications and thus prevent deterioration of the graft function. In summary, computerized prediction of relevant events has the potential to significantly change daily medical practice in patient care.

There are three major endpoints that can happen after a kidney transplantation: rejection of the kidney, graft loss and death of the patient. A rejection means that the body is rejecting the kidney. In such situation, physicians try to fight the rejection with medications and if they are not able to stop the rejection, the kidney will eventually stop working.

Our goal with this work is to predict which of these endpoints (if any) will occur to each patient 6 months and 12 months after each visit to the clinic, given the medical

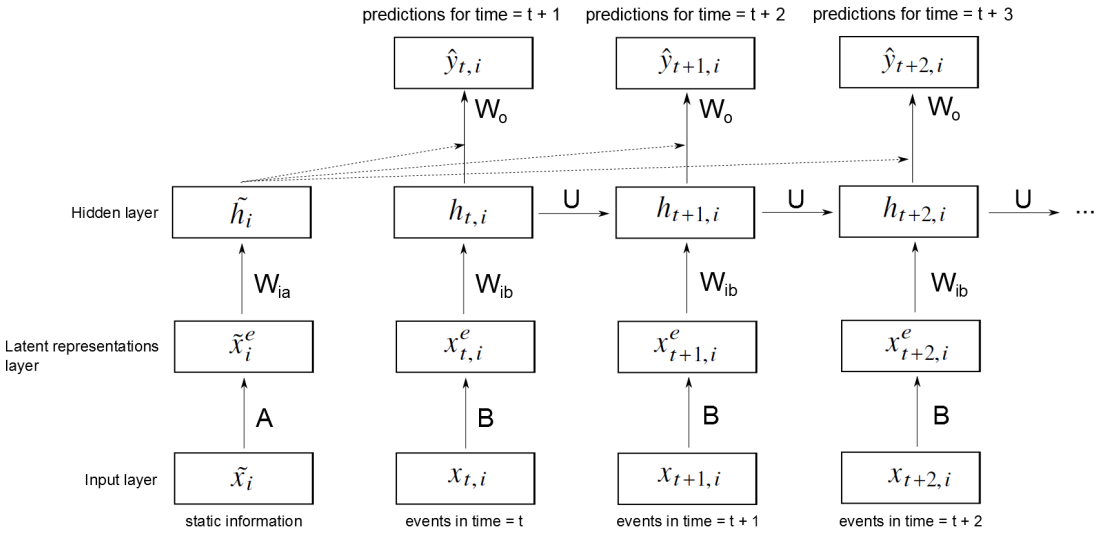

Figure 4.1: Recurrent Neural Network with static information. i stands for the index of the patient.

history of the patient. Our predictions are based on information from the patient's medical history, the sequence of medications prescribed for the patient, the sequence of laboratory tests performed together with their results, and static information, as for example age, gender, blood type, weight, primary disease, etc. Therefore each patient is represented as a sequence of events (medications prescribed and laboratory tests performed) combined with static data. We will use both static and dynamic data to predict the explained endpoints.

4.3 Recurrent Neural Networks for Clinical Event Prediction

It often happens that Electronic Health Records (EHRs) contain both dynamic information (i.e. new data is recorded every time a patient visits the clinic or hospital) and static or slowly changing information (e.g. gender, blood type, age, etc.).

Therefore, we modified the RNN architecture to include static information of the patients, such as gender, blood type, cause of the kidney failure, etc. The modified architecture is depicted in Figure 4.1.

As it can be seen, we process the static information on an independent Feedforward

Neural Network whereas we process the dynamic information with an RNN. Afterwards we concatenate the hidden states of both networks and provide this information to the output layer.

We feed our network with the latent representation of the inputs, as it is often done in Natural Language Processing [23, 93]. In this case we compute such latent representation applying a linear transformation to the raw input.

More formally, we first compute the latent representation of the input data as

$$\tilde{x}_i^e = A\tilde{x}_i \tag{4.1}$$

$$x_{t,i}^e = Bx_{t,i} \tag{4.2}$$

where $\tilde{x}_i$ is a vector containing the static information for patient i being $\tilde{x}_i^e$ its latent representation, and $x_{t,i}$ is a vector containing the information recorded during the visit made by patient i at time t, where $x_{t,i}^e$ is its latent representation.

Then we compute the hidden state of the static part of the network as

$$\tilde{h}_i = \tilde{f}(\tilde{x}_i^e) \tag{4.3}$$

where we are currently using the input and hidden layers of a Feedforward Neural Network as $\tilde{f}$.

In order to compute the hidden state of the recurrent part of the network we update:

$$h_{t,i} = f(x_{t,i}^e, h_{t-1,i}) \tag{4.4}$$

where f can be any of the update functions explained above (standard RNN, LSTM or GRU).

Finally, we use both hidden states in order to predict our target:

$$\hat{y}_{t,i} = g(\tilde{h}_i, h_{t,i}). \tag{4.5}$$

In this work we concatenate $\tilde{h}_i$ and $h_{t,i}$, and then the function g is computed as

$$g = \sigma(W_o(\tilde{h}_i, h_{t,i}) + b). \tag{4.6}$$

We derive a cost function based on the Bernoulli likelihood function, also known as Binary Cross Entropy, which has the form

$$\text{cost}(A, B, W, U) = \sum_{t,i \in \text{Tr}} -y_{t,i} \log(\hat{y}_{t,i}) - (1 - y_{t,i}) \log(1 - \hat{y}_{t,i}) \tag{4.7}$$

where Tr stands for the training dataset, $y_{t,i}$ stands for the true target data and $\hat{y}_{t,i}$ stands for the predicted data.

It is worth mentioning that we tried other architectures for combining static data with dynamic data using RNNs. For example we experimented with the approach of using an RNN to compute a latent representation of the whole history of the patient, in order to use such representation together with the latent representations of the last n visits as the inputs of a Feedforward Neural Network. We thought it could be an interesting approach since the network would put a higher attention on the most recent visits and would still have access to the full history of the patient. We also tried an attention mechanism similar to the ones shown in [21, 103], which also has the advantage of providing an interpretable model (i.e. it provides information about why a prediction was made). However none of these approaches was able to improve the performance of the models presented in this article.

4.4 Experiments

4.4.1 Data Pre-processing and Experimental Setup

We have three groups of variables in our dataset: endpoints, prescribed medications and laboratory results. Both endpoints and prescribed medications are binary data. On the other hand, the laboratory results are a set of real numbers.

Not every possible laboratory measurement is performed each time a patient visits the clinic (e.g. some times a doctor may order to measure calcium if it is suspected that the patient might have low calcium, otherwise the doctor will not order such measurement). In order to deal with such missing data, we experimented with mean imputation and median imputation. Also, we experimented with both scaling and normalizing the data before inputting it to the Neural Network.

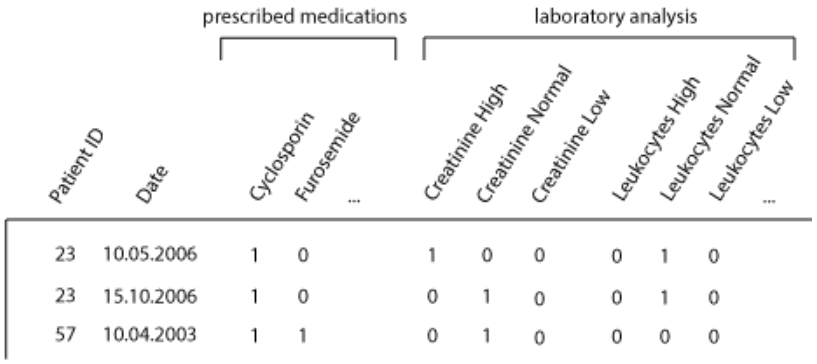

Figure 4.2: Sample of pre-processed data that we use as input for our models. Each row represents a visit to the clinic.

It turned out that encoding the laboratory measurements in a binary way by representing each of them as three event types as shown in chapter 2, i.e., LabValueHigh, LabValueNormal and LabValueLow, provided a better predictive performance than the approach of doing mean or median imputation and normalizing or scaling the data. This improvement was around 5% for the area under the ROC curve score and 3% for the area under the precision-recall curve score. In order to encode the laboratory measurements into High, Normal and Low values, we calculated the mean and standard deviation for each of them. Then, measurements greater than the mean plus the standard deviation were encoded as high, values below the mean minus the standard deviation were encoded as low, and values in between were encoded as normal. If a certain measurement was not done, its corresponding three events are all set to 0, which removes the need of imputing missing data. We believe that this improved performance provided by the discretization of the inputs, is due to the increase in the number parameters of the model and due to the fact that with this strategy we remove the need of doing data imputation, which can add a significant noise to the dataset.

The final aspect of the pre-processed data that we will use as input for our models can be found in Figure 4.2, where each row represents a visit to the clinic. The target data (i.e. the data we want to predict) will be a matrix that contains a row composed of 6 binary variables for each row on the input matrix. This 6 binary variables specify which of the 3 endpoints (if any) occurred 6 and 12 months after each visit. The possible endpoints are: kidney rejection, kidney loss and death of the patient.

Another option for performing this binary encoding would be to somehow normalize

the measured values using demographic data of each patient (e.g. gender, age, weight) or to use normal and limit values according to the medical literature. We will explore those options in future work.

In this article we consider events from the dataset that occurred on the year 2005 and onwards due to the improvement of the data quality from that year on. After cleaning the data the total number of patients is 2061, which in total have made 193111 visits to the clinic. The density of end-points (target matrix) is 7.3%, and 38.4% of the patients have suffered at least one end-point event.

The dynamic information that is generated on each visit to the clinic is composed of 1061 medications that can be prescribed and 1835 substances that can be measured in the laboratory. Thus, given that we encode each laboratory measurement into three binary variables as explained earlier, the dynamic information is composed of a total of 6566 variables. On the other hand, the static information is composed of 342 features that remain constant for each patient.

The model contains several hyperparameters that need to be optimized. The most relevant ones are the rank of the latent representations, the number of hidden units in the Neural Network, the learning rate and the dropout regularization parameter [85]. Besides we will test our models with two optimization algorithms, which are Adagrad [30] and RMSProp [94].

In order to fit these hyperparameters, we randomly split the data into three subsets: 60% of the patients were assigned to the training set, 20% of the patients were assigned to the validation set and another 20% to the test set. Under this configuration, we evaluate the performance of the model by predicting the future events of patients that the model has never seen before, and therefore increasing the difficulty of the task.

When comparing the performance of these models, we report for each model the mean area under the Precision-Recall curve (AUPRC) and mean area under Receiver Operating Characteristics curve (AUROC) together with their associated standard errors after repeating each experiment five times with different random splits of the data. We made sure that these five random splits were identical for each model.

It is worth noting that in use cases where the target event we want to predict is very infrequent, and where we are more interested in knowing when such event is going to happen (as opposed to when it is not going to happen), then the AUPRC is a more interesting score to evaluate the quality of the predictions than the AUROC. This is because getting high sensitivity and specificity can be fairly easy in these problems. However, obtaining a

high precision in the predictions is a very hard challenge. Indeed, we will show in the next section how making random predictions provides a very low AUPRC, much lower than the AUROC for random predictions. Therefore, the most interesting score to compare the models presented in this work is the AUPRC.

We will also report results on the Logistic Regression, since it provided the second best performance in the task of predicting sequences of clinical data with the Temporal Latent Embeddings shown in 2.

4.4.2 Results

Table 8.1 shows the results of predicting endpoints without considering different types of them (i.e. we concatenate all the predictions of the set and evaluate all of them together). "GRU + static", "LSTM + static" and "RNN + static" stand for the architectures presented in this chapter that combine an RNN with static information. TLE stands for the Temporal Latent Embeddings model [33]. Random stands for the scores obtained when doing random predictions.

We can see how the recurrent models outperform the other models both in the AUROC score and AUPRC score. The best performance is achieved by the GRUs with an AUPRC of 0.345 and an AUROC of 0.833. The AUPRC scores are pretty low compared to its maximum possible value (i.e. 1), but are fairly good compared to the random baseline of 0.073, which is that low due to the high sparsity of the data.

Table 4.1: Scores for full visit predictions. AUPRC stands for Area Under Precision-Recall Curve. AUROC stands for Area Under ROC Curve.

	AUPRC	AUROC
GRU + static	0.345 ± 0.013	0.833 ± 0.006
LSTM + static	0.330 ± 0.014	0.826 ± 0.006
RNN + static	0.319 ± 0.012	0.822 ± 0.006
TLE	0.313 ± 0.010	0.821 ± 0.005
Logistic Regression	0.299 ± 0.009	0.808 ± 0.005
Random	0.073 ± 0.002	0.5

Since we repeated the experiment five times with different splits of the data, we have

slight variations on the configuration of the winning model. However, the most repeated configuration was composed of 100 hidden units, a rank size of 50 for the latent representation, a dropout rate of 0.1, a learning rate of 0.1 and the Adagrad optimization algorithm.

In Table 5.3 we show the performance achieved for each specific endpoint. It can be seen how the it gets the best score for predicting the death of a patient whereas it obtains the worse AUPRC in the task of predicting kidney loss within the next 6 months.

Table 4.2: Scores for full visit predictions. AUPRC stands for Area Under Precision-Recall Curve. AUROC stands for Area Under ROC Curve.

	AUPRC	AUROC
Rejection 6 months	0.234 ± 0.010	0.778 ± 0.006
Rejection 12 months	0.279 ± 0.014	0.768 ± 0.009
Loss 6 months	0.167 ± 0.017	0.821 ± 0.009
Loss 12 months	0.223 ± 0.019	0.814 ± 0.009
Death 6 months	0.467 ± 0.018	0.890 ± 0.004
Death 12 months	0.465 ± 0.020	0.861 ± 0.004

4.4.3 Additional experiments

As we mentioned earlier, the Temporal Latent Embeddings model outperformed the baselines presented in [33] for the task of predicting the events that will be observed for each patient in his or her next visit to the clinic (i.e. which laboratory analyses will be made next, which results will be obtained in such analyses and what medications will be prescribed next). However none of those baselines were based on RNNs.

Thus we reproduced the experiments presented in [33] including the models based on RNNs introduced in this article. Table 5.4 shows the result of such experiment, where we can appreciate that the Temporal Latent Embeddings model still provides better scores than the other models. We also included in Table 5.4 an entry named “Static embeddings” which corresponds to the predictions made with a Feedforward Neural Network using just the static information of each patient. We hypothesize that the reason that Temporal Latent Embeddings model provides the best performance is due to the lack of complex long term dependencies that are relevant for this task. Thus it would be an advantage to

use a model that puts all the attention on the most recent events in situations where all the relevant information to predict the target was recorded during the previous n visits.

Table 4.3: Scores for full visit predictions. AUPRC stands for Area Under Precision-Recall Curve. AUROC stands for Area Under ROC Curve.

	AUPRC	AUROC	
TLE	0.584 ± 0.0011	0.978 ± 0.0001	
LSTM + static	0.571 ± 0.0048	0.975 ± 0.0002	
GRU + static	0.566 ± 0.0034	0.975 ± 0.0002	
Static embeddings	0.487 ± 0.0016	0.974 ± 0.0002	
Random	0.011 ± 0.0001	0.5	-

4.5 Chapter Summary

We developed and compared novel algorithms based on RNNs that are capable of combining both static and dynamic clinical variables, in order to solve the task of predicting endpoints on patients that underwent a kidney transplantation. This is an application that will provide critical information to physicians and will support them to make better decisions.

We found that an RNN with GRUs combined with a Feedforward Neural Network provides the best score for this task.

We also compared these recurrent models with other models for the task of predicting future medications and laboratory results. We found that for such use case the RNNs do not outperform a model based on a Feedforward Neural Network. We hypothesize that this is due to the lack of complex long term dependencies in the data that are relevant for this task, and therefore it is an advantage to use a model that puts all the attention to the most recent events.

We also found that binary encoding input variables that are composed of real numbers provides a better performance than normalizing the input data and performing imputation to deal with missing data.

4.6 Potential Extensions

We explained the gradient vanishing problem in Section 4.3, and later showed how LSTM and GRU units solve this problem. Interestingly, Deep Feedforward Neural Networks also suffer from this problem, also due to the several multiplications by the derivative of the sigmoid (or hyperbolic tangent) function that are accumulated in the first layers of the network. Recently there have been some efforts to solve this problem with a gating mechanism [86], similar to what is done in RNNs. However, the most common solution to solve this problem in Feedforward Neural Networks is to use Rectified Linear Units [37], whose derivative is 1 for positive input values, and therefore mitigate the gradient vanishing problem. It turns out that there have been also efforts to train standard RNNs with Rectified Linear Units in order to prevent the gradient vanishing problem, without the extra computational cost of the gating mechanisms. The results of this approach seems promising and we will add this option to our benchmark, as part of future work.

The key feature of a Clinical Decision Support System is its ability to consider as much patient information as possible and combine it in a meaningful and scalable way to predict future events. We are already capable of combining static information with a sequence of structured data. We plan to integrate more sources of data into our model with the goal of improving the quality of our predictions.

Also, having more powerful models that can deal with more data and more complex data, will allow us to tackle more difficult problems. For example, in the second experiment we predict prescribed medications, but we do not provide information regarding the doses of medications or the intake patterns. As our models improve, we will try to predict this kind of complex targets.

Finally, concerning our specific dataset composed of patients that suffered from kidney failure, there is a lot of information that we are not using yet, as for example biopsies and its results. We will keep adding to our models additional sources of information.

Chapter 5

Deep Learning for Suicidal Ideation Prediction

5.1 Mental Health and Data Analysis

According to the [101], over 800,000 people die from suicide every year[1] and the number of suicide attempts has been estimated to be 10 and 20 times this number. Suicide is the second leading cause of death in the 15-29 years age group, and its prevention is one of the top public health priorities [68]. Current suicide prevention goals include the development, implementation, and monitoring of programs that detect and prevent suicide and related behaviors.

Suicidal ideation is one type of suicidal behaviours and it is considered a major risk factor. Traditionally, suicidal ideation has been assessed by self-reports in which recall bias of autobiographical memory was present [87]. This issue can be overcome by Ecological Momentary Assesment (EMA) ([88]; [83]), which focuses on capturing symptoms and experiences at the moment they occur or very shortly thereafter, thus reducing retrospective recall biases associated with traditional medical visits. EMA also allows patients to assess themselves in their natural environments, rather than in an institutional setting, thus maximizing ecological validity. EMA has been used to investigate different psychiatric disorders and, in particular, suicidal risk [43].

Nowadays, the most common tools for implementing EMA are smartphone apps [2], in so-called Mobile Mental-Health Apps. Another term that is being used is e-Mental Health.

[1]However, since suicide is a sensitive issue, and attempts are illegal in some countries, it is very likely that incidents are under-reported.

Most of the current apps are able to track the evolution of patients but do not make use of advanced information extraction techniques; they let the clinicians interpret the raw or mild-processed data ([104]; [95]). There exists even studies based on a single patient assessment [36]. On the other hand, most of the applied Machine Learning research on suicide, from the early work of [4] to the most recent from [79], extract information from static, data-curated databases or consider single domain information like in [50]. To the best of our knowledge, there are no Machine Learning based methods tailored for making predictions from the dynamic, heterogeneous EMA data.

In our work, we make use of the dataset captured by the MEmind Wellness Tracker, an electronic EMA tool available at www.memind.net. The MEmind application has two interfaces, one for health care providers (the electronic health record view) and another for patients (the EMA view). The electronic health record view is used by physicians to store data observed during medical visits, including a standard psychiatric assessment, sociodemographic variables, diagnoses and treatments. The EMA view is used by the patients in order to submit self-reported data in real time captured in their natural environments.

In this work, we develop a Machine Learning model based on Neural Networks capable of exploiting several sources of dynamic and static information to anticipate suicidal related variables, with the ultimate goal of enabling physicians to take action when a patient is at risk of committing suicide. Our model fits perfectly well with the EMA strategy, since it is able to integrate the self-reported data captured in the natural environment of each patient with the information recorded by the doctor on each patient visit to the clinic and the background information of each patient. Besides, our approach enables us to easily integrate additional sources of information and to model complex relationships between all the variables.

There are other works in non-medical scenarios that integrate multiple sequences of data [56]. In these articles the output is another sequence instead of a value and they do not integrate static information in their models. There has been recent work aimed to predict suicides after psychiatric hospitalization in US Army soldiers [45], but in this case the variables of the study were obtained from the US administration: sociodemographic, US Army career, criminal justice, and medical or pharmacy. Therefore in this case researchers could not rely on self-reported EMA data captured within the natural environment of the patients.

5.2 MEmind Wellness Tracker Dataset

The MEmind Wellness Tracker dataset is composed of two tables. We call the first one the "EMA table" and it contains the self-reported data submitted by the patients. We call the second table the "healthcare provider table" and it contains the data collected by the physicians during the patient visits to the clinic and hospital.

5.2.1 EMA Table

The EMA table contains the information submitted by patients through the web application. It is designed to capture real-life data within the environment of the patient. This way, we obtain detailed information about the experiences of the patient while avoiding recall biases.

Patients can submit reports at any time and with the periodicity that they desire, with a maximum periodicity of one submission per day. Besides, patients are not required to answer all the questions presented in the form, although they are encouraged to do so.

Such irregularity in the submissions can make the data analysis harder to accomplish. However, the adherence to the program can also contain very useful information regarding the phenotype of patients and their actual situation.

Our dataset is currently composed of 19347 submissions made by 3016 patients. The mean number of submissions per patient is 57.02, with a median of 29 and a standard deviation of 74.88.

The web application user interface for patients consists of three sections. The first one titled "How are you today?" contains questions on appetite, sleep, happiness, anger, suicidal thoughts, adherence to treatment and the WHO-5 Well-Being Index. The WHO-5 Well-Being Index measures current mental well-being with five items asking about subjective quality of life based on positive mood, vitality and general interest rate. The second tab consists of the 12-Item General Health Questionnaire (GHQ), which is probably the most common assessment tool of mental well-being. Developed as a screening tool to detect those likely to have or to be at risk of developing psychiatric disorders, GHQ is a measure of the common mental health problems/domains of depression, anxiety, somatic symptoms and social withdrawal. Finally, there is a third tab named "Notes that was not used for this work and contains free text that the patient can use for personal notes.

All these questions are answered by the patients with numerical values between 0 and 100.

5.2.2 Healthcare Provider Table

These records are generated by the physicians when patients visit the hospital or clinic.

It is designed to capture data from standard psychiatric assessment, including sociodemographic, diagnostic and treatment information and nurse practitioner annotations (vital signs and anthropometric measurements). Some of the variables that we find in this dataset are: gender, birthdate, civil state, weight, height, heart rate, breath rate, assessment of suicidal ideation, assessment of violent behavior, environment, etc.

As it can be seen, some of this variables are constant (e.g: gender, information about about the family of the patient, etc.). Since it does not make sense to repeat this static values on every sample that belongs to certain patient, we will take this variables out of this sequential dataset and store them in a static vector.

This dataset currently contains 42262 records that correspond to 15362 patients. Each patient has on average 8.2 records with a standard deviation of 11.5.

5.3 Data preprocessing

Numeric variables were normalized to have zero mean and a standard deviation of one. Missing values were imputed with the corresponding mean value.

Categorical variables were encoded using one-hot encoding. That is, each state of a categorical variable was converted into a binary variable.

We also categorized one of the target variables. Specifically, we categorized into “high”, “medium” and “low” the suicidal ideation value reported by the patients through the web application. In order to accomplish such categorization, we calculated threshold values that left approximately the same amount of observations on each of those three buckets. Thus, samples with a reported suicidal ideation of less than 59 were classified as “low”, those with reported suicidal ideation greater than 97 were classified as “high”, and otherwise it was classified as “medium”.

5.4 Model

As we explained before, the MEmind Wellness Tracker generates two sequential datasets: one composed of submissions made by the patients through the web application, and the other one generated by the physicians containing information concerning the visits that

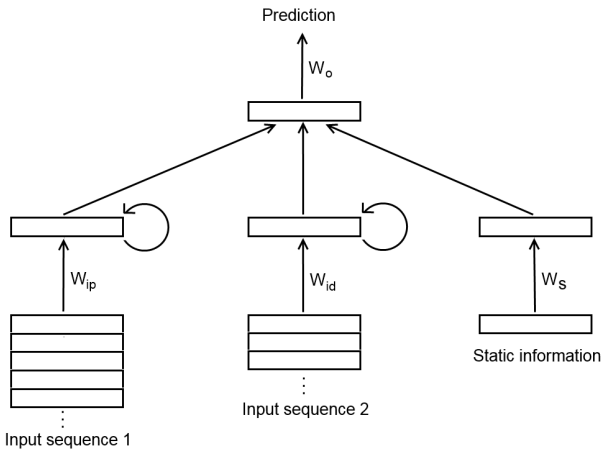

Figure 5.1: Combining multiple RNNs and static data.

each patient makes to the clinic or hospital. Indeed, due to the raise of e-health and telemedicine, it will be soon very common to have multiple sequences of data for each patient in a wide variety of medical applications.

Therefore, for this work we have developed a model that integrates these two sources of sequential data together with the static information of each patient. Besides, this approach is easily extendable to integrate additional sources of information.

5.4.1 Combining Multiple RNNs and Static Data

The model we have developed for this work is a combination of both Feedforward Neural Networks and multiple Recurrent Neural Networks. Figure 5.1 shows its architecture. It can be seen how we process each sequential dataset on an independent RNN. We also process the static data with a Feedforward Neural Network. Subsequently, we concatenate the hidden states of these two RNNs and the Feedforward Neural Network. Finally, we add an output layer in order to compute the output of the network.

More formally, we first compute the values of the input layer for each sub-network:

$$i_c^s = W_{is} x_c^s \tag{5.1}$$

$$i_{t,c}^p = W_{ip} x_{t,c}^p \tag{5.2}$$

$$i_{t,c}^d = W_{id} x_{t,c}^d \tag{5.3}$$

where x_c^s is a vector containing the static information for patient c, $x_{t,c}^p$ is a vector containing the self-reported information using the web application by patient c at time t and $x_{t,c}^d$ is a

vector containing the information recorded by the doctor during the visit made by patient c at time t. In general, s is the identifier of the sub-network that processes the static data, p is the identifier of the sub-network that processes the information submitted by the patients, and d is the identifier of the network that processes the information submitted by the doctors.W_{is}, W_{ip} and W_{ic} are matrices containing the parameters of the input layer of each sub-network.

We then compute the hidden state of each subnetwork as:

$$h_c^s = W_{hs} i_c^s \tag{5.4}$$

$$h_{t,c}^p = f_p(i_{t,c}^p, h_{t-1,c}^p) \tag{5.5}$$

$$h_{t,c}^d = f_d(i_{t,c}^d, h_{t-1,c}^d) \tag{5.6}$$

where f_p and f_p can be any of the RNN update functions explained above (standard RNN, LSTM or GRU).

Finally, we use all three hidden states in order to predict our target:

$$\hat{y}_{t,c} = g(h_c^s, h_{t,c}^p, h_{t,c}^d). \tag{5.7}$$

In this work the function g consists of passing the concatenated hidden states through an output layer:

$$g = \sigma(W_o(h_c^s, h_{t,c}^p, h_{t,c}^d) + b). \tag{5.8}$$

For regression problems we will optimize the mean squared error (MSE) whereas for classification problems we derive a cost function based on the Bernoulli likelihood function, also known as Binary Cross Entropy, which has the form:

$$\text{cost}(W, U) = \sum_{t,c \in \text{Tr}} -y_{t,c} \log(\hat{y}_{t,c}) - (1 - y_{t,c}) \log(1 - \hat{y}_{t,c}) \tag{5.9}$$

where Tr stands for the training dataset, $y_{t,c}$ stands for the true target data and $\hat{y_{t,c}}$ stands for the predicted data.

Note that even though we combine multiple sequences of data, such sequences doesn't have to be of the same length. Each RNN of the network will independently compute its hidden state by using its corresponding full sequence.

5.5 Experimental Setup

The model contains many hyperparameters that need to be optimized: number of units on the input layer of each sub-network, number of hidden units of each sub-network, learning rate and the drop out regularization parameter [85]. In order to fit these hyperparameters, we randomly split the data into three subsets: 60% of the patients were assigned to the training set, 20% of the patients were assigned to the validation set and another 20% to the test set. Therefore we evaluate the performance of the model by predicting the future events of patients that the model has never seen before, which increases the difficulty of the task.

In order to calculate the confidence intervals of the reported scores, we will train each model with 5 different random splits of the data. We will made sure that these 5 splits are identical for all models. In order to train the models, we will use the Adagrad algorithm [31] and we will report the mean value of the scores achieved and their associated standard error.

As part of the static information, we will include the number of days since the last report. Therefore, we will be conditioning each prediction to the number of days between the date of the predicted report and the date of the last report observed.

We perform each experiment with our proposed model using both GRUs and LSTM units.

We also include as benchmark Logistic Regression applied to the concatenated last samples of each sequence together with the static information of each patient. Finally, we include a naive prediction that consists of just predicting as next suicidal ideation value the reported suicidal ideation in the previous submission.

5.6 Results

5.6.1 Predicting Suicidal Ideation

Our first experiment consists of predicting the suicidal ideation value that will be reported in each submission. In other words, for each submission available, we try to anticipate which suicidal ideation level will be observed next.

Table 8.2 shows the mean squared error (MSE) for each model. It can be seen how our model using GRU units ("GRUs + static") performs best for this problem. "LSTMs + static" represent our model using LSTM units. "Predicting mean" represents the MSE

achieved when we just always predict the mean suicidal ideation value.

Table 5.1: Mean squared error for suicidal ideation prediction.

	Mean Squared Error
GRUs + static	0.0340 ± 0.0004
LSTMs + static	0.0357 ± 0.0004
Logistic Regression	0.0372 ± 0.0007
Predicting mean	0.2248 ± 0.0091

The architecture of the winning model varies slightly among the different splits of the data, but the most common configuration is composed of 25 GRU units on each RNN, 10 units to process the static information, and 500 units in the output layer. Furthermore, 0.01 was the best value found for the drop-out regularization parameter and 0.01 was also the best value found for the learning rate.

5.6.2 Predicting Suicidal Ideation Level

As explained in Section 5.3, we categorized the suicidal ideation reported by the patients into "high", "medium" and "low" as part of the web application. Our task consists of predicting which of these states will be observed in the next submission that each patient makes. In other words, for each submission available, we try to anticipate which suicidal ideation level will be observed next.

This categorization of the suicidal ideation will allow physicians to define customized thresholds to, for example, get an automatic alarm when some patient gets into the high suicidal ideation level. Also, it allows us to evaluate our models with classification scores such as Area under the ROC curve (AUROC) and Area under the Precision-Recall curve (AUPRC). Note that the AUPRC for random predictions decreases with the sparsity of the data, meaning that achieving a high AUPRC is harder for sparse datasets. However, AUPRC is the most interesting score when you are also interested in the precision of the predictions. We also report the BCE (Binary Cross Entropy) error, which is the cost (the lower the better) that we are optimizing during training.

Table 8.1 shows the BCE cost, AUPRC and AUROC for the combined predictions of the three classes. It can be seen how our model using GRU units performs best for this

problem. "Repeating previous" represents the scores achieved when we just predict that the future reported ideation level will be the same than the present suicidal ideation level.

Table 5.2: Combined scores. BCE the lower the better.

	BCE	AUPRC	AUROC
GRUs + static	0.265 ± 0.010	0.903 ± 0.007	0.945 ± 0.004
LSTMs + static	0.279 ± 0.010	0.892 ± 0.007	0.941 ± 0.004
Logistic Regression	0.296 ± 0.009	0.858 ± 0.007	0.924 ± 0.004
Repeating previous	1.517 ± 0.007	0.882 ± 0.005	0.894 ± 0.005
Random	1	0.333 ± 0.001	0.5

In this case, the most common configuration is composed of 50 GRU units on each RNN, 10 units to process the static information, and 150 units in the output layer. Besides, 0.01 was the best value found for the drop-out regularization parameter and 0.01 was also the best value found for the learning rate.

We show also the specific scores achieved for each predicted class: low suicidal ideation, medium suicidal ideation and high suicidal ideation. Table 5.3 shows the AUPRC for each class, whereas 5.4 shows the AUROC for each class.

Table 5.3: Area under Precision-Recall curve class wise (Suicidal ideation low, medium and high).

	AUPRC Low	AUPRC Medium	AUPRC High
GRUs + static	0.872 ± 0.011	0.871 ± 0.006	0.940 ± 0.005
LSTMs + static	0.866 ± 0.009	0.856 ± 0.008	0.931 ± 0.008
Logistic Regression	0.830 ± 0.025	0.794 ± 0.006	0.897 ± 0.080
Repeating previous	0.862 ± 0.005	0.863 ± 0.004	0.920 ± 0.008
Random	0.306 ± 0.001	0.359 ± 0.004	0.332 ± 0.001

Table 5.4: Area under the ROC curve class wise (Suicidal ideation low, medium and high).

	AUROC Low	AUROC Medium	AUROC High
GRUs + static	0.943 ± 0.004	0.922 ± 0.005	0.963 ± 0.004
LSTMs + static	0.940 ± 0.005	0.913 ± 0.006	0.959 ± 0.004
Logistic Regression	0.926 ± 0.006	0.893 ± 0.005	0.939 ± 0.006
Repeating previous	0.888 ± 0.004	0.867 ± 0.004	0.927 ± 0.005
Random	0.5	0.5	0.5

5.7 Sensitivity analysis

We performed a univariate sensitivity analysis in order to find out which input variables contain more information for the task of predicting the next suicidal ideation value. Table 5.5 shows the relevance score for the 10 most informative variables.

Table 5.5: 10 most informative variables to predict suicidal ideation.

Variable	Score
suicidal_ideation	1.1973
apetite_hyporexia	1.1113
apetite_regular	1.0956
apetite_hyperexia	1.0936
apetite	1.0893
sleep_hypersomnia (>8.4h)	1.0887
sleep_regular (6-8.4h)	1.0847
sleep_quality_regular	1.0840
agressivity_no_agressivity	1.0792
sleep_insomnia (<6h)	1.0776
who_5	1.0737
ghqsStress	1.0726

The relevance score associated to each variable is computed as the ratio between the MSE achieved when such variable was remove and the less informative variable. Thus,

when suicidal_ideation variable is discarded, the MSE to predict the next suicidal ideation is 1.19 times higher than when the less informative variable is removed. It can be seen that the present time is the most informative variable to predict the suicidal ideation in the future. It is followed by the variables related to appetite and the ones related to sleep. The relationship between suicidal behaviour and eating ([92]) and sleeping patterns ([16];[89]; [98]) has been been studied multiple times before.

5.8 Chapter Summary

We have developed a model capable of aggregating multiples sequences of data and static data. Thus, it is a great fit for the EMA approach followed by physicians in the field of psychiatry. We compared different versions of the proposed model and included Logistic Regression as benchmark. Our main result is that the version of our model using GRU units provides the best performance. More importantly, we have presented a framework specifically suited for sequential data that can be extended with additional sources of data.

Finally, even though Neural Networks are considered "black box models" that provide no insight, we have performed a sensitivity analysis which shows how, aside the present suicidal ideation, the appetite and sleep amount are the most informative variables to predict future suicidal ideation. This finding supports the current observations made by the doctors in their daily practice.

Chapter 6

Deployment

So far we have described the theoretical challenges that we had to overcome in order to use Deep Learning to model medical datasets. However this thesis pursues a very pragmatical goal, which is helping doctors to provide a better care of their patients. For this reason, it is essential to actually integrate the this kind of algorithms into healthcare institutions so that physicians can take advantage of Machine Learning models on their daily activities.

In this chapter we will provide a description of the software developed in this thesis to deploy the one of our models into a server located at the Charité University Hospital in Berlin as a prototype clinical decision support system.

The model selected to be deployed is the endpoint prediction algorithm explained in Chapter 4. We chose to deploy this model for several reasons: the quality of its predictions is good enough to be tried in the clinic, the fact that the output prediction is composed of just 6 numbers makes it easier to be visualized and understood by the doctors, and it is a system that could provide very relevant information whereas in the clinic there is currently no other system that provides similar information.

The system was programmed in the Python programming Language, using the Deep Learning framework named Theano[1] combined with Python scientific libraries such us numpy[44], scipy[44], pandas[59], etc.

The deployed system is composed of two main parts: the front end and the back end. The back end is where the trained model is located, whereas the front end contains the user interface that physicians can use to make predict future endpoints. Figure 6.3 shows a high level schema of the architecture of the system.

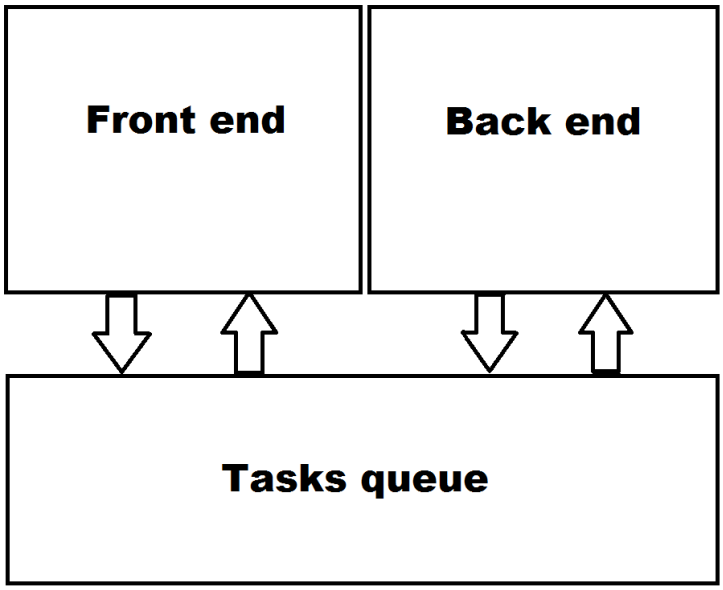

Figure 6.1: Architecture of the clinical decision support system.

6.1 The task queue

The task queue is a directory where tasks are stored. A task is a prediction that a physician has put on the queue, and they can be in one of the following states: waiting to be processed, in process, or completed. Each of these tasks is stored in the form of a directory within the task queue folder. Each task's directory contains the set of CSV (comma separated value) files need for computing such prediction. Specifically, in order to compute the endpoints predictions, we make use of the following five tables of the hospital's database: patients, transplantations, medications, ordered laboratory analysis and laboratory analysis results. Thus, each task folder will contain five CSV files, each of them composed of the rows associated to the patient for which we want to compute the endpoint predictions.

6.2 The front end

The first step in the prediction process is to provide physicians with a user interface where they can introduce the ID of the patient for which they would like to compute the endpoint predictions.

For this purpose, we programmed a web user interface and set up a web server using Python, a library named Flask for website generation and serving, and the Tornado web

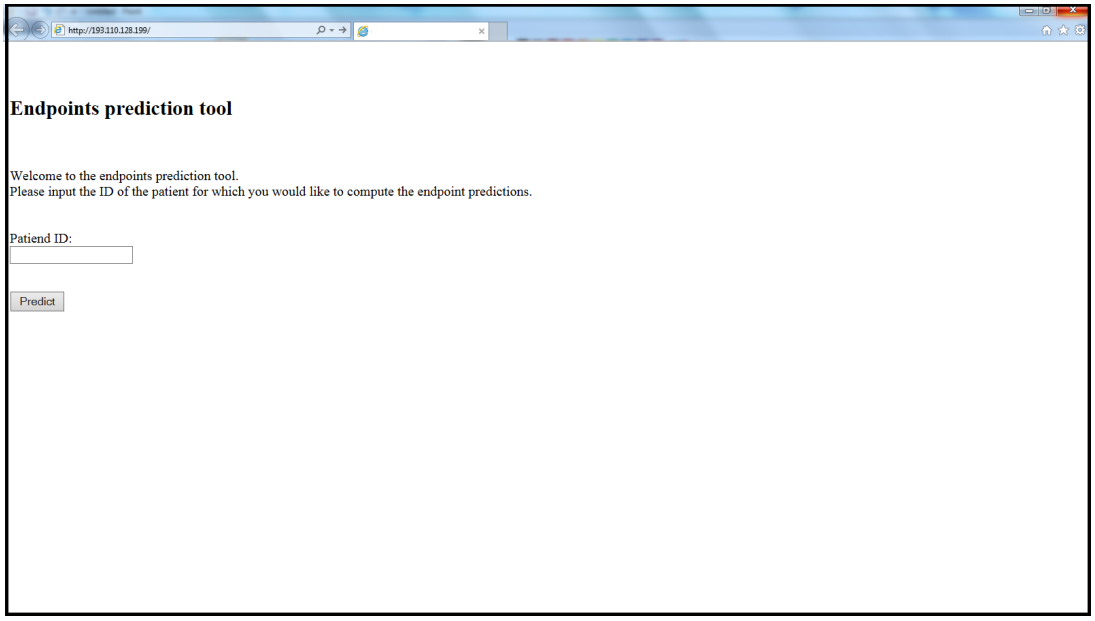

Figure 6.2: Web user interface for endpoint prediction.

server. In this website, which is only accessible from within the Charité intranet, doctors find a form where they input the ID of the patient whose endpoints prediction they would like to obtain. Finally, they click on the "Predict" button and, after a couple of seconds, a table displaying the predictions for each of the six endpoints will appear in the web browser.

Internally, once a doctor clicks on the "Predict" button, there is a Python script that connects to the database, retrieves the rows of the corresponding patient in the five tables of interest, and stores them as CSV files. These five files are stored inside a new folder within the task queue directory. In order to indicate that all CSV files have been saved successfully and the task is ready to be processed, the script in the front end also generates an empty file named "ready_for_predictor".

After the "ready_for_predictor" file has been generated, the back end will continue with the process and eventually it will generate a "predictions.json" file containing the output of the computation. Thus, the front end will wait until such "predictions.json" file is generated, and once it detects such file, it will load the predictions and render a website showing them to the doctor that launched the prediction. To make the system easier to maintain, the whole back end is stored in a single Python program that can be easily replaced or updated.

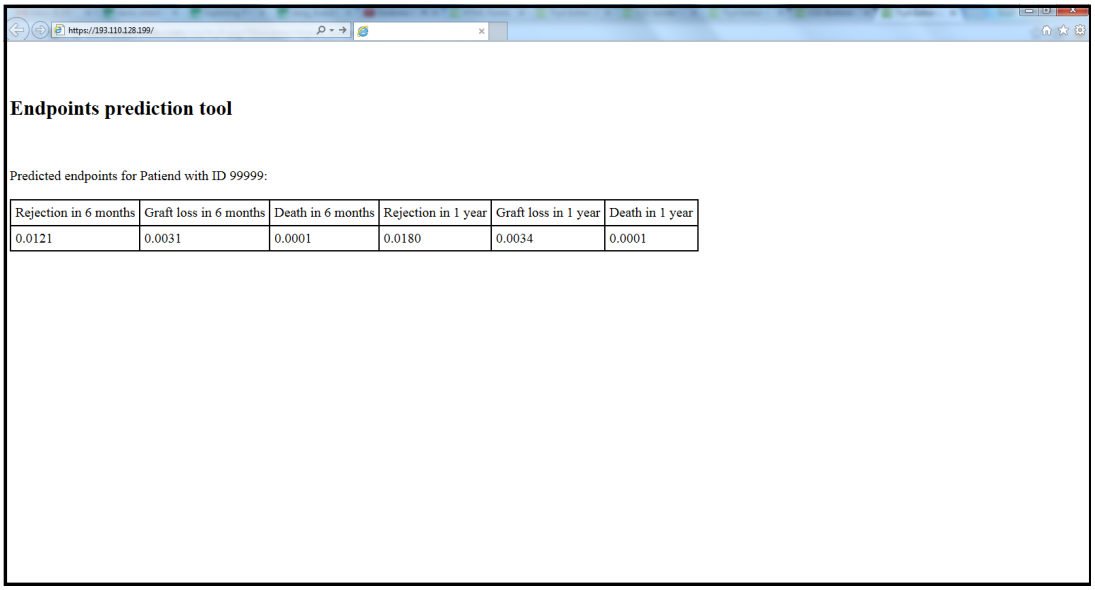

Figure 6.3: Hypothetical results for a kidney transplantation patient.

6.3 The back end

The back end is the part of the system in charge of running the Machine Learning model in order to compute the predictions.

For now, the system works only with pre-trained models. Therefore we must first train the model before using in the clinic. For this purpose, the back end contains both a JSON (JavaScript Object Notation) file describing the architecture of the model (number of layers, units in each layer, etc.) and an additional file containing the weights of the model. Therefore, as soon as the back end is started, it will load the JSON files, then it will instantiate a Neural Network with the loaded parameters, and finally it will load the weights of the network.

Once the model is loaded in memory, the back end will permanently monitor the task queue, in order to detect when a new task has been located in the queue. When it finds a task where the "ready_for_predictor" file is present, it will load the data from such task, compute the prediction, and store the results in a JSON file named "prediction.json", which contains the probability of each endpoint happening in the next 6 months and 1 year.

Chapter 7

Generative Adversarial Networks for Medical Decision Modelling

If we use Neural Networks and backpropagation to model processes that present multiple mutually exclusive valid outcomes, they learn to predict the average of such valid outcomes. There are applications for which this is not the desired behavior. For example, let's imagine that we use a Neural Network in order to color black and white images. Given a good training set, the Neural Network will probably learn to correctly color objects such as mountains or human faces, since the model can find features on those objects that are highly correlated with their colors. However, what about cars? The color of a car is mostly completely independent of its shape, brand, model, etc. Thus, the network will learn that any color is valid for any car. It will subsequently predict a high probability for every color and all cars will end up appearing grey in the pictures produced by our model.

The same issue occurs in medical data. Lets suppose we train a Neural Network for predicting the medications that will be prescribed in each situation. For every patient, the model will yield a high probability for each potential medication that could be prescribed. For example, if the patient needs an antibiotic, the model will probably yield a high probability for every antibiotic in the database that is regularly prescribed for that disease. Thus, this set of predictions cannot be considered a correct prescription, since the model is suggesting to prescribe many equivalent medications. Therefore, this model can be understood as recommender system whose predictions need to be interpreted by a human expert that subsequently generates the right prescription.

However, Neural Networks are the most suitable Machine Learning models for modelling complex processes. Thus, efforts have been made to overcome the aforementioned

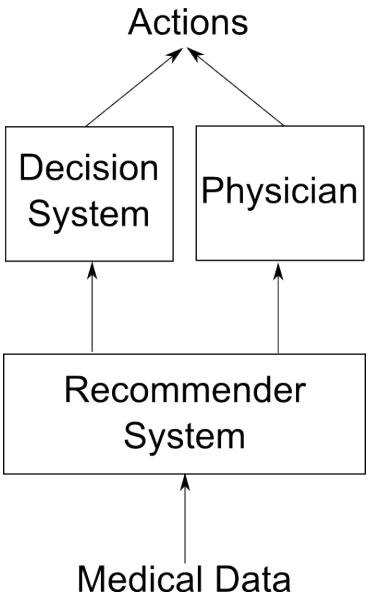

Figure 7.1: The recommendations generated by a Neural Network need to be interpreted by a physician. In this chapter we will present an additional model that can play the role of the physician and turn the recommendations into valid decisions.

problem and to be able to use them to generate data that looks realistic. One of the most relevant models that have been developed for this purpose are Generative Adversarial Networks (GANs). Our goal in this section is to analyze how GANs can be used in order to create a decision layer that works on top of the recommender systems we built in previous chapters. This decision layer will model the behavior of the human decision maker that takes recommendations as inputs and then decides which action to take. The schema of the whole system is depicted in Figure 7.1.

In order to experiment with these models in a controlled environment, we will generate a synthetic dataset with which we will run our experiments. Real data will be of course much more challenging than the simulated one, but running experiments on synthetic data is the first step that needs to be taken before moving to a real scenario.

7.1 Generative Adversarial Neural Networks

GANs are a type of Neural Network architecture introduced in [38] which is specifically designed to build generative models. GANs are composed of two elements: a generator

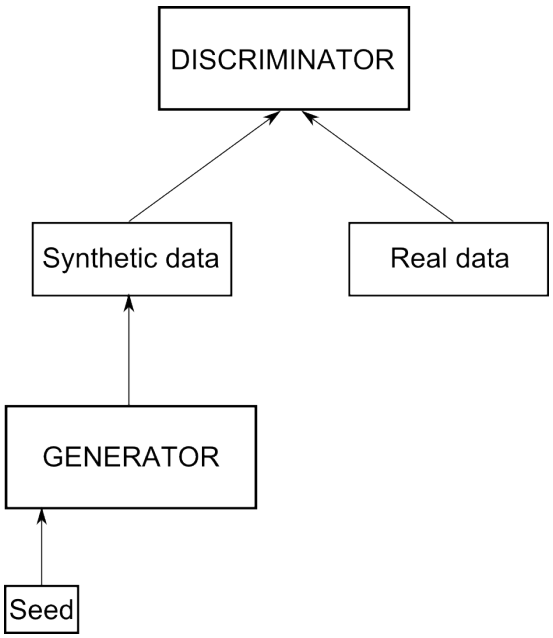

Figure 7.2: Architecture of a Generative Adversarial Network. The discriminator is trained to classify if the data samples are real or synthetic. The generator tries to generate synthetic data that looks realistic.

and a discriminator.

The generator is an Neural Network that can have any kind of internal architecture. Its goal is to generate synthetic samples of data that look like the ones collected in the real world. The generator receives a randomly initialized vector as input, which is the source of randomness that enables it to generate different samples of data by using the same internal parameters. On the other hand, the discriminator is a binary classifier whose goal is to distinguish between samples collected in the real world and samples generated by the generator. The GAN architecture is depicted in Figure 7.2

The learning process consists of these two elements playing a sort of game against each other. The discriminator tries to learn to differentiate real world samples from the synthetic samples generated by the generator. On the other hand, the generator can see what strategy the discriminator is following to tell the difference between real and synthetic samples. The generator subsequently modifies its parameters in order to generate samples that trick the classifier into label them as real. This training process is explained in more detail in the next paragraphs.

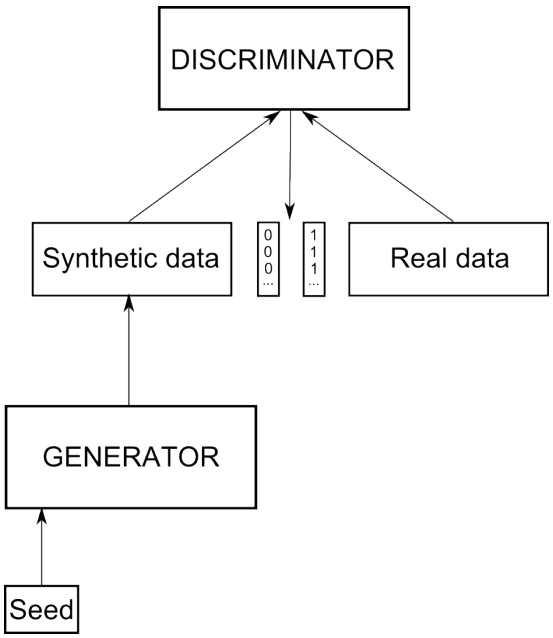

Figure 7.3: During the training of the discriminator, synthetic samples are labeled with a "0", whereas real samples are labeled with a "1".

7.1.1 Discriminator Training

As it was previously mentioned, both the generator and the discriminator can have any internal NN architecture, which normally depends on the type of data that we want to generate, and its parameters are randomly initialized as in any other NN model.

The learning process can start either by training the discriminator or the generator. In order to train the discriminator, we first generate a set of synthetic data with our generator. Ideally we will generate as many samples as we have in our real dataset, which is the data we are trying to replicate, so that the discriminator can see the same number of samples of each type. Afterwards, we label each synthetic data sample with a "0" and each real data sample with a "1", as depicted in Figure 7.3.

Once the data is labeled, we merge both datasets (synthetic data and real data), and we train the discriminator. The discriminator is just a binary classifier that will learn to assign a "0" to the synthetic data and a "1" to the real data. Therefore, during this part of the process, we temporarily ignore the generator, as we show in Figure 7.4.

More formally, the discriminator is trained to minimize the average negative cross-entropy between its predictions and the labels of the data. If we denote by NN(X) the

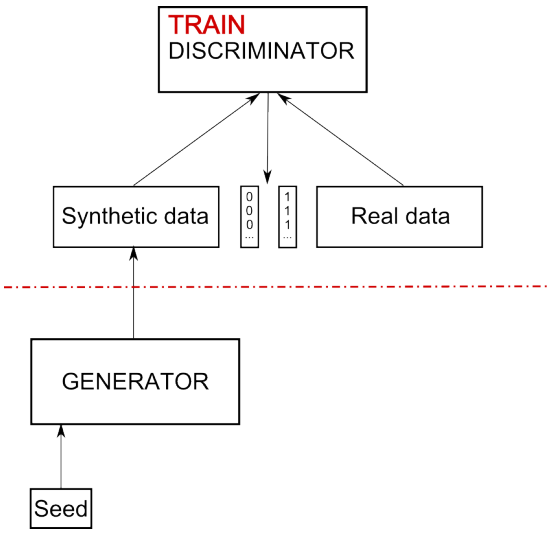

Figure 7.4: During the training of the discriminator, the generator is ignored.

vector or matrix comprising outputs from a NN receiving a vector or matrix X and by $\mathrm{CE}(A, B)$ the averaged cross-entropy between two matrices A and B , then the discriminator loss for a pair $\{X_n, y_n\}$ (with $X_n \in \mathbb{R}^d$ and $y_n \in \{1, 0\}$) is:

$$\mathrm{D}_{\mathrm{loss}}(X_n, y_n) = -\mathrm{CE}(\mathrm{NN}_{\mathrm{D}}(X_n), y_n)$$

When X is composed of synthetic inputs, y_n is a vector of 0s, while it is a vector of 1s when X is composed of real inputs. In each training mini-batch, the discriminator sees both real and synthetic inputs.

After training the discriminator we proceed to train the generator. In the experiments that will be presented in this chapter, we train the generator after training the discriminator with one mini-batch of 128 samples.

7.1.2 Generator Training

In order to train the generator, we ignore the real data. The goal in this part of the process is to train the generator to generate samples that the discriminator labels as "1". Note that we previously trained the classifier to label real samples as "1" and synthetic ones as "0". Therefore, we now train the generator to generate samples that are difficult to differentiate from real data.

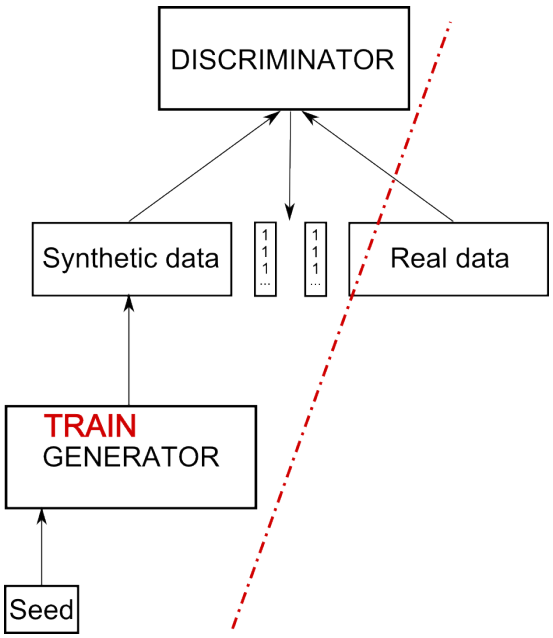

Figure 7.5: During the training of the generator, we try to adapt its internal parameters so that the discriminator labels its generated samples as real.

In order to do this, we consider the generator plus the discriminator as one single network. We train this network to always predict "1"s, but we only update the parameters of the generator. The optimization process is carried out as with any other Neural Network using stochastic gradient descent. Essentially, during this part of the process, we are finding the parameters of the generator that make its generated samples be classified as real data by the discriminator. This process is depicted in Figure 7.5.

More formally, the generator tries to minimize the following cost function:

$$\mathrm{G}_{\mathrm{loss}}(Z_n) = \mathrm{D}_{\mathrm{loss}}(\mathrm{NN}_{\mathrm{G}}(Z_n), 1) = -\mathrm{CE}(\mathrm{NN}_{\mathrm{D}}(\mathrm{NN}_{\mathrm{G}}(Z_n)), 1)$$

Here Z_n is a set of vectors sampled independently from the latent/noise space Z, thus $Z_n \in \mathbb{R}^m$ since $Z = \mathbb{R}^m$.

After training the generator with this strategy, we go back to train the discriminator. We can keep iterating between both for a fixed number of epochs or until the generated samples look real enough. As shown in [38], it can be shown that this process is equivalent to minimizing the Jensen-Shannon divergence between the real distribution of the data and the synthetic distribution.

7.2 Conditional Generative Adversarial Networks for Medication Prescriptions

Some of the models presented earlier in this thesis play the role of recommender systems and their outcome is a vector that contains the probability of each medication being prescribed. As we explained earlier in this chapter, such vectors are not always valid prescriptions. For example, there can be a situation were multiple medications are valid but just one can be prescribed. In that case, providing a high probability to each valid medication cannot be considered as a valid prescription, but as a recommendation that some human expert has to interpret. The human expert can then use the recommendations provided to issue a valid prescription. Our goal in this chapter is to add a new layer, on top of the recommender system, aimed to model the behavior of the human decision maker. Thus, the decision layer will take as input a vector containing a recommendation for medication prescriptions, and it will generate another vector with a valid prescription.

However, the GAN architecture shown in Section 7.1 takes only two inputs: the data we want to replicate, and a random seed. Therefore, this architecture does not enable the possibility of conditioning the generated data to some additional input. Since we need our GAN to generate a valid prescription given a specific recommendation, we will use the conditional version of the GAN, as presented in [62]. Figure 7.6 shows how we use the conditional GAN in order to convert a set of recommendations into valid medication prescriptions. As represented in the figure, medication probabilities provided by an external recommendation system are fed both to the generator and to the discriminator. By inputting this information to the generator, we are asking it to generate a valid prescription for the provided recommendation vector. On the other hand, by providing this information to the discriminator, we are letting it know that the prescription that it is trying to classify should correspond to the recommendations provided.

As suggested in [62], the conditioning input is appended to each layer of both the discriminator and the generator. This process is depicted in Figure 7.7, where the conditioning input is labeled as “input 2”.

7.3 Synthetic Medication Prescription Dataset

There is a fundamental difficulty that prevents us from training generative models for medical decisions, which is that when multiple options are equally correct for certain decision

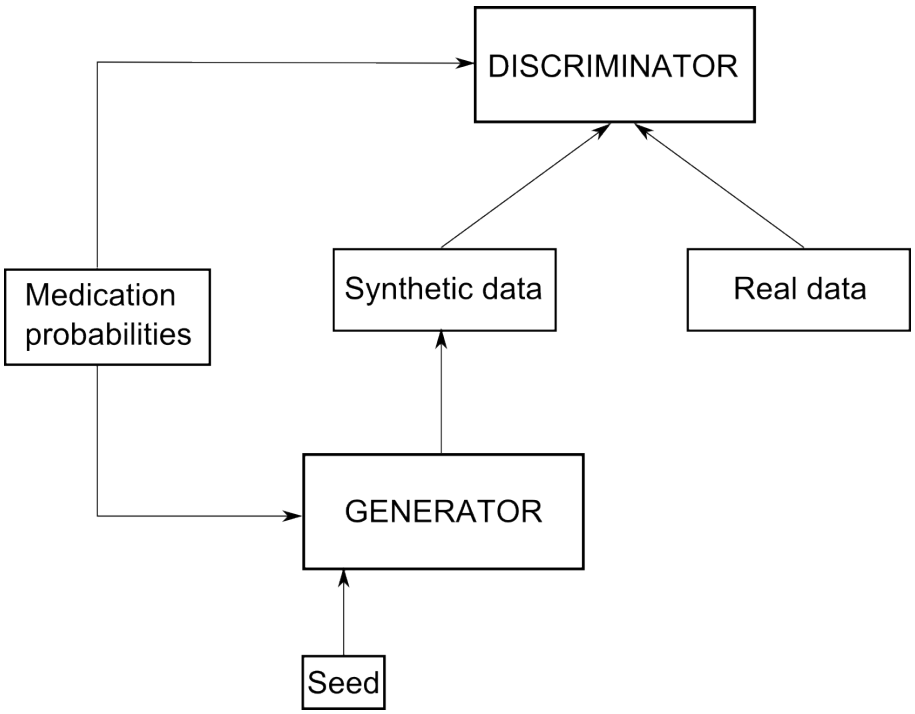

Figure 7.6: Conditional GAN arquitecture to turn a set of recommendations into valid medication prescriptions.

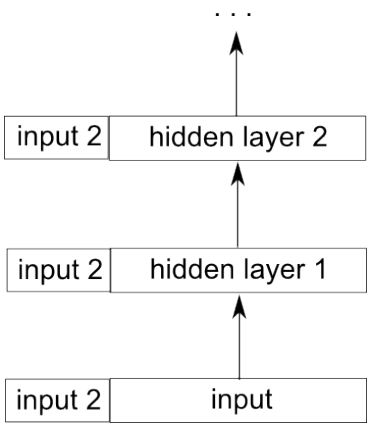

Figure 7.7: Conditional GAN architecture.

process, our ground truth will only indicate one of them. For example, lets imagine that for a given situation, a patient can be prescribed with any of three medications, being all of them equally suitable for his or her disease. The physician will prescribe one of the medications and such decision will be stored in our dataset. Afterwards, if we use our

model to replicate such decision process, it may happen that our model prescribes a correct medication for the situation, but still a different one than the doctor prescribed. In such scenario, when we evaluate the prediction made by our model, it will be considered as a failed prediction, since our ground truth indicates that the right medication is a different one.

Thus, in order to be able to correctly model the prescription generation process, our ground truth must include all the correct options for every sample in the dataset. However, if we are working with real medical data, getting such complete ground truth can be really hard to get, if not impossible. No hospital or clinic is currently storing all possible correct decisions for each situation. Thus, getting a complete ground truth would involve getting help from physicians in order to label each sample with all the correct decisions that could have been taken.

For this reason, we decided to generate a simplified dataset, where we simulate a set of diseases each of which can be treated with multiple medications. This simulated dataset consists of the information described in the following subsections.

7.3.1 Diseases Vector

It is a randomly generated vector of length d that is composed of integer numbers, where each position represents a disease, and the integers represent the number of medications that are equally appropriated to treat each disease. For example, the vector $[3, 2, 4]$ would represent a dataset where there are 3 possible diseases, and the number of medications to treat each disease are 3, 2 and 4 respectively.

For our experiments, we use a vector length d equal to 5 which we randomly initialize following a Gaussian distribution with mean 10 and standard deviation 2. The specific diseases vector used in our experiments is the following: $[9.0, 10.0, 6.0, 13.0, 6.0]$. Therefore, there will be a total of 44 medications in our dataset.

Such disease vector does not look as one that can be find in real datasets very often. However, it has been designed to clearly show the issues that can arise when non-generative models are used to generate medication prescriptions. It has been also designed to be able to prove this point with a small dataset where Neural Network models can be trained in a short period of time.

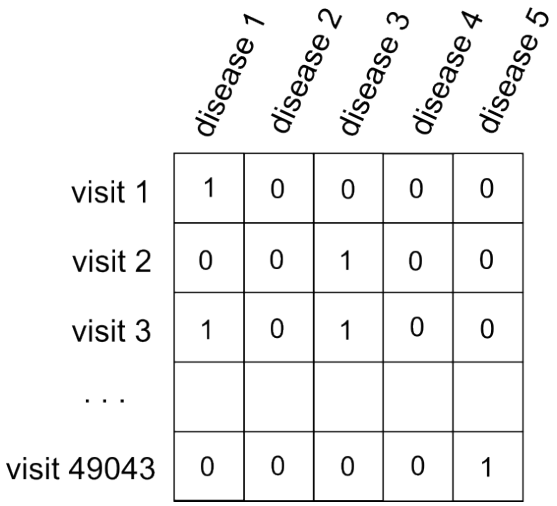

Figure 7.8: Diagnosis matrix sample.

7.3.2 Diagnosis Matrix

This is a binary matrix with size $v \times d$, where each row represents a visit to the clinic and each column corresponds to one of the diseases available in the dataset. For example, a 1 located in the position $(2, 4)$ would indicate that during the second recorded visit a physician diagnosed the diseases number 4 to the patient.

For our experiments, in order to create this matrix, we first generate a sparse matrix with 1 million rows and a sparsity of 0.01. Afterwards, we remove the rows that contain no 1s, since we assume that every visit contains at least a diagnosed disease. After running this procedure, we kept 49043 rows in our diagnosis matrix. Since we are working with 5 potential diseases, the shape of our diagnosis matrix is 49043×5. Figure 7.8 shows a sample of this matrix.

7.3.3 Medications Prescriptions Matrix

The medications prescription matrix contains a simulation of the medication that a doctor would prescribe for each visit, assuming he or she always chooses one of the correct medications for each disease. This is a binary matrix with size $v \times m$, where each row represents a visit to the clinic and each column corresponds to one of the medications available in the dataset.

In order to simulate such decision process, we randomly select one of the valid medications for each disease present on each visit.

Figure 7.9: Medications prescriptions matrix sample.

For our experiments we simulated 49043 visits and 44 medications, therefore the medications prescription matrix will have a size of 49043×44. Figure 7.9 shows a sample of this matrix.

7.4 Experiments

7.4.1 Training the Recommender System

Since we want to simulate the whole system depicted in Figure 7.1, we start by using a regular Feedforward NN that serves as a recommender system. This NN will take the diagnosis matrix as input and will predict the values of the medication prescription matrix. In order words, it will predict the medications prescribed for each visit given the observed diseases. Thus, it has 5 inputs, which is the number of diseases in our dataset, and 44 outputs, which is the number of medications in our dataset. We will use 80% of the samples in our dataset for training, 20% for validation and 20% for test. We train it using Adagrad[30] and perform early stopping using the validation set.

After the training, we evaluate the model by using the test set, and we obtain an Area under the ROC curve (AUROC) of 0.923 and an Area under the Precision-Recall curve (AUPRC) of 0.158. The AUPRC for random predictions is 0.0231.

However, we know that the prescriptions contained in the medication prescription ma-

trix, are not the only valid choice for each visit. Therefore, we developed a method that dynamically takes the ground truth that will generate the best AUROC and AUPRC scores for each prediction. That is, among all the right prescriptions that could be issued to a patient, we evaluate our model with the prescription that will provide the best score for the predictions provided by the model. We will refer to this evaluation method as "complete ground truth". Note that the main reason for using simulated data is to be able to perform this kind of evaluation, since using real data it is hard to obtain all the medication prescriptions that would have been valid on each visit.

After applying the "complete ground truth" evaluation strategy, we achieve an Area under the ROC curve (AUROC) of 0.958 and an Area under the Precision-Recall curve of 0.506. The AUPRC for random predictions is 0.0233.

7.4.2 Training the GAN

The next step is to take the outcome of this NN and use it to train a GAN. Following the architecture presented in Figure 7.6, the "Real data" box is the data we are trying to mimic, in our case the Medication Prescription Matrix. On the other hand, the "Medication probabilities" box contains the probabilities generated by the recommender system. In order words, we want to train a generator that, given the output probabilities of the Feedforward NN, generates a valid prescription as a physician would do. Both the generator and the discriminator are Feedforward Neural Networks with two hidden layers following the architecture shown in 7.7.

The generator takes as input a randomly initialized vector and the vector of probabilities produced by the recommender system, which in our case will be composed of 44 real numbers, since that is the number of medications available in our dataset. The output of the generator, will be another vector with 44 real numbers, representing the probability of each medication being prescribed. Thus, as we explained before, the goal of the generator is to transform the recommendations provided by a regular NN into valid prescriptions, as depicted in Figure 7.1.

After each epoch, we use the "complete ground truth" evaluation method and the validation set in order to check the quality of the prescription generated by the generator. We stop the training when the validation error of the prescriptions doesn't improve for seven consecutive epochs. Concerning the hyperparameters, the configuration that provided the best results had a generator composed of 100 units in the first hidden layer and 10 units in the second one. The discriminator was composed of 10 hidden units in the first hidden

layer and 500 in the second one. For both networks the best learning rate was 0.05 and the L2 regularization parameter was 0.1. Besides, the random seed vector used in the discriminator had a length of 2, and the optimization algorithm used to train the whole system was Adagrad.

After training, the generator achieved an AUROC of 0.990 and AUPRC of 0.704 on the test set. Thus, it can be seen how the GAN actually improved the performance of the predictions provided by the first NN. In other words, the decision layer is actually taking the probabilities of the recommender system and converting them to more valid prescriptions. Table 7.1 summarizes all the results obtained in the experiment.

Table 7.1: AUROC stands for Area Under ROC Curve. AUPRC stands for Area Under Precision-Recall Curve. FNN stands for Feedforward Neural Network. CGT stands for Complete Ground Truth. GAN stands for Generative Adversarial Networks.

	AUROC	AUPRC
FNN	0.923	0.158
FNN CGT	0.958	0.506
GAN CGT	0.990	0.704

7.5 Conclusion and Future Work

In this chapter we took a first step towards building fully autonomous clinical decision systems. We created a synthetic dataset to run our experiments and showed how a GAN architecture can be used to build a decision layer than improves the performance of a regular NN.

There is currently a lot of research going on the field of generative models based on deep learning architectures. It will be interesting to see how the new methods perform in our synthetic dataset. Eventually, these methods must be tested with real data, since it will be a much more complex task than modelling synthetic data. However, this task will require to find a dataset where all potential valid actions are known.

Chapter 8

Predicting the Co-Evolution of Event and Knowledge Graphs

8.1 Data Modelling with Knowledge Graphs

In this chapter, we present an approach using knowledge graphs (KG) that can be considered a general case of the models presented earlier in this thesis, and which can be applied to other domains aside from healthcare. In previous publications it was shown how a KG composed of triplets can be represented as a multiway array (tensor) and how a statistical model can be formed by deriving latent representations of generalized entities. Successful models are, e.g., RESCAL [67], Translational Embeddings Models [18] and the multiway neural networks as used in [29]. In these publications KGs were treated as static: A KG grew more links when more facts became available but the ground truth value associated with a link was considered time invariant. In this chapter we address the issue of KGs where triple states depend on time. In the simplest case this might consider simple facts like "Obama is president of the United States", but only from 2008-2016. Another example is a patient whose status changes from sick to healthy or vice versa. Most popular KGs like Yago [90], DBpedia [3] Freebase [17] and the Google Knowledge Graph [84] have means to store temporal information.

Without loss of generality, we assume that changes in the KG always arrive in form of events, in the sense that the events are the gateway to the KG. For a given time step, events are described by a typically very sparse event triple graph, which contains facts that change some of the triples in the KG, e.g., from *True* to *False* and vice versa. KG triples which do not appear in the event graph are assumed unchanged.

An example might be the statement that a patient has a new diagnosis of diabetes, which is an information that first appears as an event in the event graph but is then also transmitted to the KG. Other events might be a prescription of a medication to lower the cholesterol level, the decision to measure the cholesterol level and the measurement result of the cholesterol level; so events can be, e.g., actions, decisions and measurements. In a similar way as the KG is represented as a KG tensor, the event graphs for all time steps can be represented as an event tensor. Statistical models for both the KG tensor and the event tensor can be derived based on latent representations derived from the tensor contents.

Although the event tensor has a representation for time, it is by itself not a prediction model. Thus, we train a separate prediction model which estimates future events based on the latent representations of previous events in the event tensor and the latent representations of the involved generalized entities in the KG tensor. In this way, a prediction can, e.g., use both background information describing the status of a patient and can consider recent events. Since some future events will be absorbed into the KG, by predicting future events, we also predict likely changes in the KG and thus obtain a model for the evolution of the KG as well.

There is a wide range of papers on the application of data mining and machine learning to KGs. Data mining attempts to find interesting KG patterns [11, 77, 73]. Some machine learning approaches attempt to extract close-to deterministic dependencies and ontological constructs [58, 35, 52]. This chapter focuses on *statistical* machine learning in KG where representation learning has been proven to be very successful.

There is considerable prior work on the application of tensor models to temporal data, e.g., EEG data, and overviews can be found in [46] and [63]. In that work, prediction is typically not in focus, but instead one attempts to understand essential underlying temporal processes by analyzing the derived latent representations.

Some models consider a temporal parameter drift. Examples are the BPTF [102], and [32]. Our model has a more expressive dynamic by explicitly considering recent histories. Markov properties in tensor models were considered in [75, 76]. In that work quadratic interactions between latent representations were considered. The approach described here is more general and also considers multiway neural networks as flexible function approximators.

Our approach can also be related to the neural probabilistic language model [8], which coined the term *representation learning*. It can be considered an event model where the occurrence of a word is predicted based on most recent observed words using a neural

network model with word representations as inputs. In our approach we consider that several events might be observed at a time instance and we consider a richer family of latent factor representations.

There is considerable recent work on dynamic graphs [53, 69, 91, 78] with a strong focus on the Web graph and social graphs. That work is not immediately applicable to KGs but we plan to explore potential links as part of our future work.

8.2 The Knowledge Graph Model

With the advent of the Semantic Web [14], Linked Open Data [13], Knowledge Graphs (KGs) [90, 17, 84], triple-oriented knowledge representations have gained in popularity. Here we consider a slight extension to the subject-predicate-object triple form by adding the value $(e_s, e_p, e_o;$ *Value*$)$ where *Value* is a function of s, p, o and can be the truth value of the triple or it can be a measurement. Thus *(Jack, likes, Mary; True)* states that Jack likes Mary, and *(Jack, hasBloodTest, Cholesterol; 160)* would indicate a particular blood cholesterol level for Jack. Note that e_s and e_o represent the entities for subject index s and object index o. To simplify notation we also consider e_p to be a generalized entity associated with predicate type with index p.

A machine learning approach to inductive inference in KGs is based on the factor analysis of its adjacency tensor $\underline{X}$ where the tensor element $x_{s,p,o}$ is the associated *Value* of the triple (e_s, e_p, e_o). Here $s = 1, \ldots, S$, $p = 1, \ldots, P$, and $o = 1, \ldots, O$. One can also define a second tensor $\underline{\Theta}^{KG}$ with the same dimensions as $\underline{X}$. It contains the natural parameters of the model and the connection to $\underline{X}$. In the binary case one can use a Bernoulli likelihood with $P(x_{s,p,o}|\theta^{KG}_{s,p,o}) \sim \text{sig}(\theta^{KG}_{s,p,o})$, where $\text{sig}(arg) = 1/(1 + \exp(-arg))$ is the logistic function. If $x_{s,p,o}$ is a real number than we can use a Gaussian distribution with $P(x_{s,p,o}|\theta^{KG}_{s,p,o}) \sim \mathcal{N}(\theta^{KG}_{s,p,o}, \sigma^2)$.

In representation learning, one assigns an r-dimensional latent vector to the entity e denoted by $\mathbf{a}_e = (a_{e,0}, a_{e,1}, \ldots, a_{e,r})^T$. We then model using one function

$$\theta^{KG}_{s,p,o} = f^{KG}(\mathbf{a}_{e_s}, \mathbf{a}_{e_p}, \mathbf{a}_{e_o})$$

or, using one function for each predicate,

$$\theta^{KG}_{s,p,o} = f^{KG}_p(\mathbf{a}_{e_s}, \mathbf{a}_{e_o}).$$

For example, the RESCAL model [67] is

$$\theta^{KG}_{s,p,o} = \sum_{k=1}^{r}\sum_{l=1}^{r} R_{p,k,l} a_{e_s,k} a_{e_o,l},$$

where $R \in \mathbb{R}^{P\times r\times r}$ is the core tensor. In the multiway neural network model [29] one uses

$$\theta^{KG}_{s,p,o} = \mathrm{NN}(\mathbf{a}_{e_s}, \mathbf{a}_{e_p}, \mathbf{a}_{e_o})$$

where NN stands for a neural network and where the inputs are concatenated. These approaches have been used very successfully to model large KGs, such as the Yago KG, the DBpedia KG and parts of the Google KG. It has been shown experimentally that models using latent factors perform well in these high-dimensional and highly sparse domains. For a recent review, please consult [66].

We also consider an alternative representation. The idea is that the latent vector stands for the tensor entries associated with the corresponding entity. As an example, $\mathbf{a}_{e_s}$ is the latent representation for all values associated with entity e_s, i.e., $x_{s,:,:}$. [1] It is then convenient to assume that one can calculate a so-called M-map of the form

$$\mathbf{a}_{e_s} = M^{subject} x_{s,:,:}. \tag{8.1}$$

Here $M^{subject} \in \mathbb{R}^{r\times(PO)}$ is a mapping matrix to be learned and $x_{s,:,:}$ is a column vector of size PO.[2] For multilinear models it can be shown that such a representation is always possible; for other models this is a constraint on the latent factor representation. The advantage now is that the latent representations of an entity can be calculated in one simple vector matrix product, even for new entities not considered in training. We can define similar maps for all latent factors. For a given latent representation we can either learn the latent factors directly, or we learn an M-matrix.

The latent factors, the M-matrices, and the parameters in the functions can be trained with penalized log-likelihood cost functions described in the Appendix.

8.3 The Event Model

Without loss of generality, we assume that changes in the KG always arrive in form of events, in the sense that the events are the gateway to the KG. For a given time step,

[1] If an entity can also appear as an object ($o : e_o = e_s$), we need to include $x_{:,:,o}$.

[2] The M matrices are dense but one dimension is small (r), so in our settings we did not run into storage problems. Initial experiments indicate that random projections can be used in case that computer memory becomes a limitation.

events are described by a typically very sparse event triple graph, which contains facts that change some of the triples in the KG, e.g., from *True* to *False* and vice versa. KG triples which do not appear in the event graph are assumed unchanged.

Events might be, e.g., *do a cholesterol measurement*, the event *cholesterol measurement*, which specifies the value or the order *take cholesterol lowering medicine*, which determines that a particular medication is prescribed followed by dosage information.

At each time step events form triples which form a sparse triple graph and which specifies which facts become available. The event tensor is a four-way tensor $\underline{Z}$ with $(e_s, e_p, e_o, e_t; \mathit{Value})$ and tensor elements $z_{s,p,o,t}$. We have introduced the generalized entity e_t to represent time. Note that the characteristics of the KG tensor and the event tensor are quite different. $\underline{X}$ is sparse and entries rarely change with time. $\underline{Z}$ is even sparser and nonzero entries typically "appear" more random. We model

$$\theta^{event}_{s,p,o} = f^{event}(\mathbf{a}_{e_s}, \mathbf{a}_{e_p}, \mathbf{a}_{e_o}, \mathbf{a}_{e_t}).$$

Here, $\mathbf{a}_{e_t}$ is the latent representation of the generalized entity e_t.

Alternatively, we consider a personalized representation of the form

$$\theta^{pers\text{-}event,s=i}_{p,o} = f^{pers\text{-}event}(\mathbf{a}_{e_p}, \mathbf{a}_{e_o}, \mathbf{a}_{e_s=i,t}).$$

Here, we have introduced the generalized entity $e_{s,t}$ for a subject $s = i$ at time t which stands for all events of entity $s = i$ at time t.

Since representations involving time need to be calculated online, we use M-maps of the form

$$\mathbf{a}_{e_{s,t}} = M^{subject,\ time} z_{s,:,:,t}$$

The cost functions are again described in the Appendix.

8.4 The Prediction Model

8.4.1 Predicting Events

Note that both the KG-tensor and the event tensor can only model information that was observed until time t but it would not be easy to derive predictions for future events, which would be of interest, e.g., for decision support. The key idea of this chapter is that events are predicted using both latent representations of the KG and latent representations describing recently observed events.

In the prediction model we estimate future entries in the event tensor $\underline{Z}$. The general form is

$$\theta^{predict}_{s,p,o,t} = f^{predict}(args) \quad \text{or} \quad \theta^{predict}_{s,p,o,t} = f^{predict}_{p,o}(args)$$

where the first version uses a single function and the latter uses a different function for each (p, o)-pair.[3] Here, *args* is from the sets of latent representations from the KG tensor and the event tensor.

An example of a prediction model is

$$\theta^{predict}_{s,p,o,t} = f^{predict}_{p,o}(\mathbf{a}_{e_s}, \mathbf{a}_{e_{s,t}}, \mathbf{a}_{e_{s,t-1}}, \ldots, \mathbf{a}_{e_{s,t-T}}).$$

where the prediction is based on the latent representations of subject, object and predicate from the KG-tensor and of the time-specific representations from the event tensor.

Let's consider an example. Let $(e_s, e_p, e_o, e_t; \mathit{Value})$ stand for (*Patient, prescription, CholesterolMedication, Time; True*). Here, $\mathbf{a}_{e_s}$ is the profile of the patient, calculated from the KG model. Being constant, $\mathbf{a}_{e_s}$ assumes the role of parameters in the prediction model. $\mathbf{a}_{e_{s,t}}$ describes all that *so far* has happened to the patient at the same instance in time t (e.g., on the same day). $\mathbf{a}_{e_{s,t-1}}$ describes all that happened to the patient at the last instance in time and so on.

We model the functions by a multiway neural network with weight parameters W exploiting the great modeling flexibility of neural networks. The cost function for the prediction model is

$$\begin{aligned} \text{cost}^{predict} = &- \sum_{z_{s,p,o,t} \in \underline{Z}} \log P(z_{s,p,o,t} | \theta^{predict}_{s,p,o,t}(A, M, W)) \\ &+ \lambda_A \|A\|_F^2 + \lambda_W \|W\|_F^2 + \lambda_M \|M\|_F^2. \end{aligned} \tag{8.2}$$

A stands for the parameters in latent representation and M stands for the parameters in the M-matrices. For a generalized entity for which we use an M-matrix, we penalize the entries in the M-matrix; for a generalized entity for which we directly estimate the latent representation we penalize the entries in the corresponding latent terms in A. Here, $\|\cdot\|_F$ is the Frobenius norm and $\lambda_A \geq 0$, $\lambda_M \geq 0$ and $\lambda_W \geq 0$ are regularization parameters.

8.4.2 Predicting Changes in the KG

In our model, each change in the status of the KG is communicated via events. Thus each change in the KG first appears in the event tensor and predictions of events also implies

[3] The different functions can be realized by the multiple outputs of a neural network.

predictions in the KG. The events that change the KG status are transferred into the KG and the latent representations of the KG, i.e., $\mathbf{a}_{e_s}, \mathbf{a}_{e_p}, \mathbf{a}_{e_o}$, are re-estimated regularly (Figure 8.1).

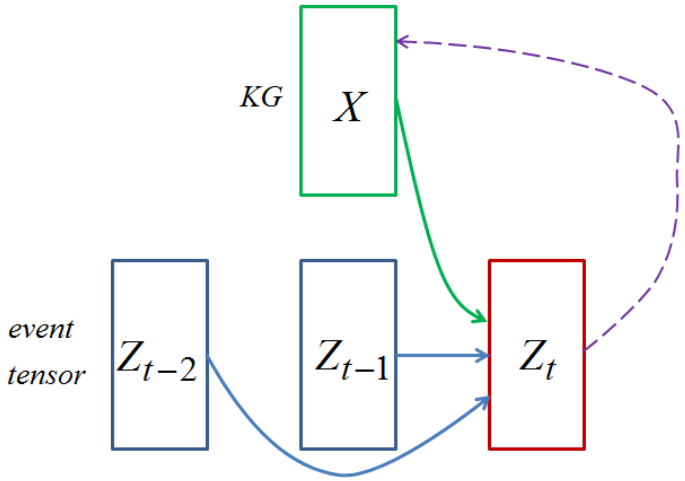

Figure 8.1: The figure shows an example where the event tensor is predicted from the representations of the events in the last two time steps and from the KG representation. The dotted line indicate the transfer of observed events into the KG.

8.4.3 More Cost Functions

Associated with each tensor model and prediction model, there is a cost function (see Appendix). In our experiments we obtained best results, when we used the cost function of the task we are trying to solve. In the most relevant prediction task we thus use the cost function in Equation 8.2. On the other hand, we obtained faster convergence for the prediction model if we initialize latent representations based on the KG model.

8.5 Experiments

8.5.1 Modelling Clinical Data

The study is based on the same dataset as chaperts 2 and 4. It is a large dataset collected from patients that suffered from kidney failure. As explained earlier, the dataset contains

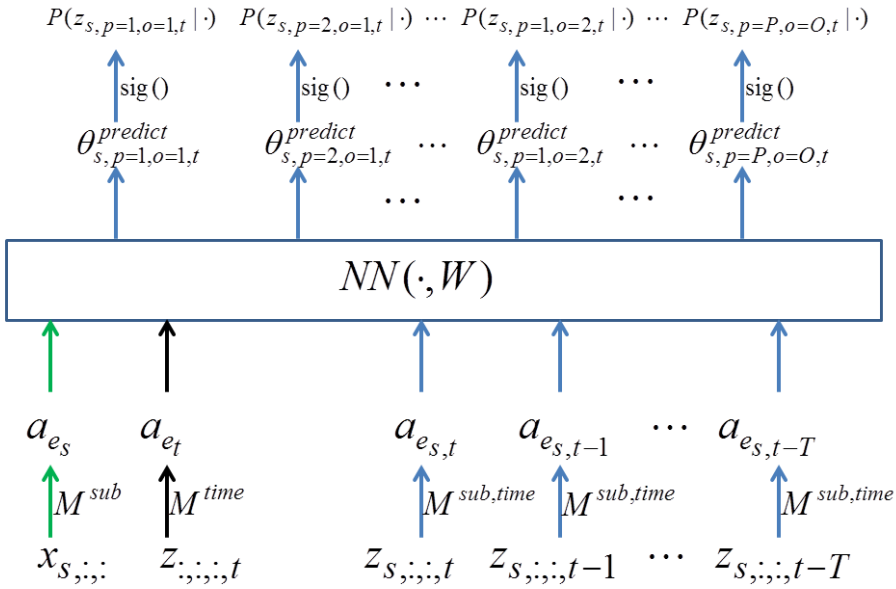

Figure 8.2: The prediction model for the clinical data.

Table 8.1: Scores for next visit predictions. AUPRC stands for Area Under Precision-Recall Curve. AUROC stands for Area Under ROC Curve. *ET* stands for our proposed model that uses only past event information but no information from the KG.

	AUPRC	AUROC	Time (hours)
ET	0.574 ± 0.0014	0.977 ± 0.0001	6.11
Logistic Regression	0.554 ± 0.0020	0.970 ± 0.0005	4.31
KNN	0.482 ± 0.0012	0.951 ± 0.0002	17.74
Naive Bayes	0.432 ± 0.0019	0.843 ± 0.0015	39.1
Constant predictions	0.350 ± 0.0011	0.964 ± 0.0001	0.001
Random	0.011 ± 0.0001	0.5	-

every event that happened to each patient concerning the kidney failure and all its associated events: prescribed medications, hospitalizations, diagnoses, laboratory tests, etc. [54, 82]. It is composed of dozens of tables with more than 4000 patients that underwent a

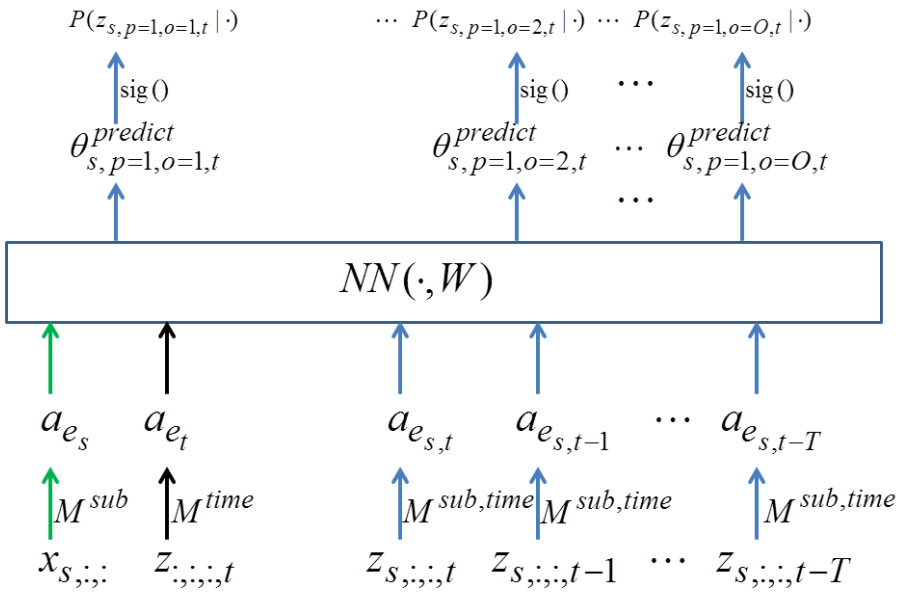

Figure 8.3: The prediction model for the recommendation data.

renal transplant or are waiting for it. For example, the database contains more than 1200 medications that have been prescribed more than 250000 times, and the results of more than 450000 laboratory analyses.

This is particularly important for the estimation of drug-drug interactions (DDI) and adverse drug reactions (ADR) in patients after renal transplant.

We work with a subset of the variables available in the dataset. Specifically, we model medication prescriptions, ordered lab tests and lab test results. We transformed the tables into an event oriented representation where the subject is the patient and where time is a patient visit. Also in this chapter, we encoded the lab results in a binary format representing normal, high, and low values of a lab measurement, thus *Value* is always binary.

The prediction model is

$$\theta^{predict}_{s,p,o,t} = f^{predict}_{p,o}(\mathbf{a}_{e_s}, \mathbf{a}_{e_t}, \mathbf{a}_{e_{s,t}}, \mathbf{a}_{e_{s,t-1}}, \ldots, \mathbf{a}_{e_{s,t-T}}).$$

Note that we have a separate function for each (p, o)-pair. $\mathbf{a}_{e_s}$ are patient properties as described in the KG. $\mathbf{a}_{e_{s,t}}$ represents all events known that happened at visit t for the

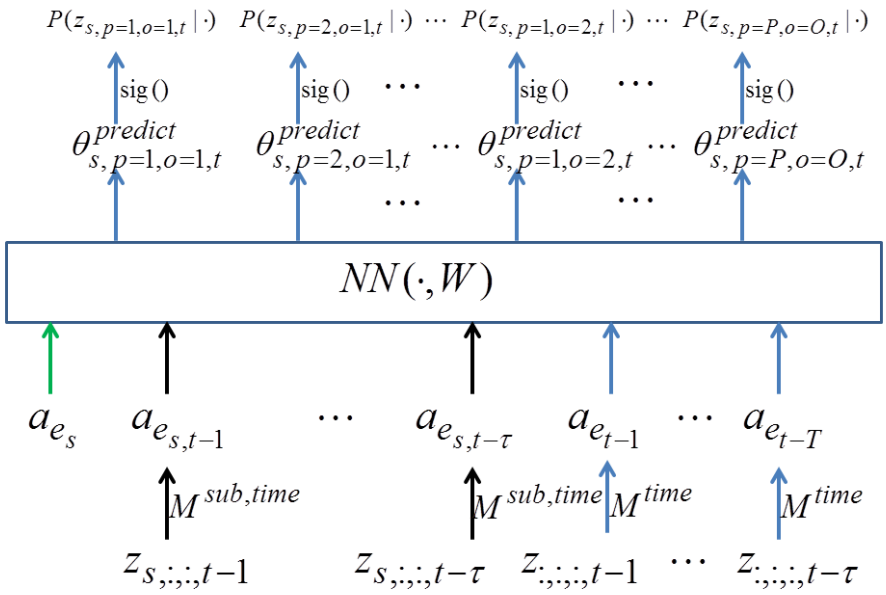

Figure 8.4: The prediction model for the sensor data. $\mathbf{a}_{e_s}$ is directly estimated without using an M-mapping.

patient (e.g., the same visit for which we want to make a prediction). $\mathbf{a}_{e_{s,t-1}}$ represents all events for the patient at the last visit, etc. $\mathbf{a}_{e_t}$ stands for the latent representation of all events at visit t for all patients and can model if events are explicitly dependent on the time since the transplant. Regarding the input window, we empirically found that $T = 6$ is optimal. The architecture is shown in Figure 8.2.

The first experiment consisted of predicting the events that will happen to patients in their next visit to the clinic given the events that were observed in the patients' previous visits to the clinic (i.e. by using the events that occurred to the patient from $\mathbf{a}_{e_{s,t}}$ until $\mathbf{a}_{e_{s,t-6}}$). The experiment was performed 10 times with different random splits of the patients. Thus we truly predict performance on patients which were not considered in training! Table 8.1 shows how our proposed model outperforms the baseline models. The "constant predictor" always predicts for each event the occurrence rate of such event (thus the most common event is given the highest probability of happening, followed by the second most common event, and so on). Note that we are particularly interested in the Area Under Precision-Recall Curve score due to the high sparsity of the data and our interest

in predicting events that will actually happen, as opposed to the task of predicting which events will not be observed. In the last column of Table I we also report the time that it took to train for each model with the best set of hyperparameters in the first random split.

Next we repeat the experiment including the KG-representation of the patient, which contains static variables of the patient such as blood type and gender, i.e., $\mathbf{a}_{e_s}$, and also used $\mathbf{a}_{e_t}$. Table 8.2 shows the improvement brought by the inclusion of the KG representation. The last row in Table 8.2 shows the result of making the predictions just with the KG representation of the patient (i.e. without the past event information), demonstrating clearly that information on past events is necessary to achieve best performance.

Table 8.2: Scores for full visit predictions with and without the information in the KG. AUPRC stands for Area Under Precision-Recall Curve. AUROC stands for Area Under ROC Curve. *ET+KG* stands for our proposed model that uses past event information and information from the KG. *ET* only uses past event data and *KG* only uses KG data.

	AUPRC	AUROC
ET+KG	0.586 ± 0.0010	0.979 ± 0.0001
ET	0.574 ± 0.0014	0.977 ± 0.0001
KG	0.487 ± 0.0016	0.974 ± 0.0002

8.5.2 Recommendation Engines

We used data from the MovieLens project with 943 users and 1682 movies.[4] In the KG tensor we considered the triples *(User, rates, Movie; Rating)*. For the event tensor, we considered the quadruples *(User, watches, Movie, Time; Watched)* and *(User, rates, Movie, Time; Rating)*. Here, $Rating \in \{1, \ldots, 5\}$ is the score the user assigned to the movie and $Watched \in \{0, 1\}$ indicates if the movie was watched and rated at time t. *Time* is the calendar week of the rating event. We define our training data to be 78176 events in the first 24 calender weeks and the test data to be 2664 events in the last 7 weeks. Note that in both datasets there are only 738 users since the remaining 205 users watched and rated their movies only in the test set.

[4] http://grouplens.org/datasets/movielens/

It turned out that the movie ratings did not show dependencies on past events, so they could be predicted from the KG model alone with

$$\theta^{predict}_{s,rates,o,t} = f^{predict}_{rates,o}(\mathbf{a}_{e_s}).$$

We obtained best results by modeling the function with a neural network with 1682 outputs (one for each movie). The user specific data was centered w.r.t. to their average and a numerical 0 would stand for a neutral *rating*. We obtain an RMSE score of 0.90 ± 0.002 which is competitive with the best reported score of 0.89 on this dataset [75]. But note that we predicted *future ratings* which is more difficult than predicting randomly chosen test ratings, as done in the other studies. Since we predict ordinal ratings, we used a Gaussian likelihood model.

Of more interest in this chapter is to predict if a user will decide to watch a movie at the next time step. We used a prediction model with

$$\theta^{predict}_{s,watches,o,t} =$$
$$f^{predict}_{watches,o}(\mathbf{a}_{e_s}, \mathbf{a}_{e_t}, \mathbf{a}_{e_{s,\ t}}, \mathbf{a}_{e_{s,t-1}}, \ldots, \mathbf{a}_{e_{s,t-T}}).$$

Here, $\mathbf{a}_{e_s}$ stands for the profile of the user as represented in the KG. $\mathbf{a}_{e_t}$ stands for the latent representation of all events at time t and can model seasonal preferences for movies. $\mathbf{a}_{e_{s,t}}$ stands for the latent representation of all movies that the user watched at time t. The architecture is shown in Figure 8.3. When training with only the prediction cost function we observe an AUROC an score of 0.728 ± 0.001. We then explored sharing of statistical strength by optimizing jointly the M-matrices using all three cost functions cost^{KG}, cost^{event} and $\text{cost}^{predict}$ and obtained a significant improvement with an AUROC score of 0.776 ± 0.002.

For comparison, we considered a pure KG-model and achieved an AUROC score of 0.756 ± 0.007. Thus the information on past events leads to a small (but significant) improvement.

8.5.3 Sensor Networks

In our third experiment we wanted to explore if our approach is also applicable to data from sensor networks. The main difference is now that the event tensor becomes a sensor tensor with subsymbolic measurements at all sensors at all times.

Important research issues for wind energy systems concern the accurate wind profile prediction, as it plays an important role in planning and designing of wind farms. Due to

Table 8.3: Mean Squared Error scores for predicting multivariate sensor data 20 time steps ahead.

Model	MSE
Pred1	0.135 ± 0.0002
Pred2	0.139 ± 0.0002
Pred3	0.137 ± 0.0002
Feedforward Neural Network	0.140 ± 0.0002
Linear Regression	0.141 ± 0.0001
Last Observed Value	0.170

the complex intersections among large-scale geometrical parameters such as surface conditions, pressure, temperature, wind speed and wind direction, wind forecasting has been considered a very challenging task. In our analysis we used data from the Automated Surface Observing System (ASOS) units that are operated and controlled cooperatively in the United States by the NWS, FAA and DOD[5]. We downloaded the data from the Iowa Environmental Mesonet (IEM)[6]. The data consists of 18 weather stations (the *Entities*) distributed in the central US, which provide measurements every minute. The measurements we considered are wind strength, wind direction, temperature, air pressure, dew point and visibility coefficient (the *Attributes*).

In the analysis we used data from 5 months from April 2008 to August 2008. The original database consists of 18 tables one for each station.

The event tensor is now a sensor tensor with quadruples *(Station, measurement, SensorType, Time; Value)*, where *Value* is the sensor measurement for sensor *SensorType* at station *Station* at time *Time*. The KG-tensor is a long-term memory and maintains a track record of sensor measurement history.

As the dataset contains missing values we only considered the periods in which the data is complete. This results in a total of 130442 time steps for our dataset. In order to capture important patterns in the data and to reduce noise, we applied moving average smoothing using a Hanning window of 21 time steps. We split the data into train-, validation- and test set. The first four months of the dataset where used for training, and the last month

[5]http://www.nws.noaa.gov/asos/

[6]https://mesonet.agron.iastate.edu/request/asos/1min.phtml

as test set. 5 % of the training data where used for validation.

We considered three different prediction models with Gaussian likelihood functions, each with different latent representations at the input. The first model (Pred1) is

$$\theta^{predict}_{s,p,o,t} = f^{predict}_{p,o}(\mathbf{a}_{e_s}, \mathbf{a}_{e_{s,t-1}}, \mathbf{a}_{e_{s,t-2}}, \ldots, \mathbf{a}_{e_{s,t-T}})$$

where $\mathbf{a}_{e_{s,t}}$ stands for all measurements of station e_s at time t and $\mathbf{a}_{e_{s,t-1}}, \mathbf{a}_{e_{s,t-2}}, \ldots, \mathbf{a}_{e_{s,t-T}}$ can be considered a short term memory. $\mathbf{a}_{e_{s,t-1}}$ represents all measurements for station s between $t-T-1$ and $t-1$, i.e., and can represent complex sensor patterns over a longer period in time. Since measurements take on real values, a Gaussian likelihood model was used.

The second model (Pred2) is

$$\theta^{predict}_{s,p,o,t} = f^{predict}_{p,o}(\mathbf{a}_{e_{s,t-1}}, \ldots, \mathbf{a}_{e_{s,t-T}}, \mathbf{a}_{e_{t-1}}, \ldots, \mathbf{a}_{e_{t-T}}).$$

Here, $\mathbf{a}_{e_t}$ stands the latent representation of all measurements in the complete network at time t.

And finally the third model (Pred3) combines the first two models and uses the combined sets of inputs. The architecture of Pred3 is shown in Figure 8.4.

In our experiments we considered the task of predicting 20 time steps into the future. All three models performed best with $T = 10$ and the rank of the latent representations being 20. Table 8.3 summarizes the results of the three prediction models together with three baseline models. The most basic baseline is to use the last observed value of each time series as a prediction. More enhanced baseline models are linear regression and feedforward neural networks using the previous history $z_{s,:,:,t-1}, z_{s,:,:,t-2}, \ldots, z_{s,:,:,t-T}$ of all time series of a station s as input. The experiments show that all three prediction models outperform the baselines. Pred1, which adds the personalization term for each sensor shows the best results. Pred2 performs only slightly better than the feedforward neural network. However, we assume that in sensor networks with a stronger cross correlation between the sensors, this model might prove its strength. Finally, the result of Pred3 shows that the combination of the multiple latent representations is too complex and does not outperform Pred1.

8.6 Conclusions and Extensions

We have introduced an approach for modeling the temporal evolution of knowledge graphs and for the evolution of associated events and signals. We have demonstrated experimentally that models using latent representations perform well in these high-dimensional

and highly sparse dynamic domains in a clinical application, a recommendation engine and a sensor network application. The clinical application is explored further in a funded project [34]. As part of future work we plan to test our approach in general streaming frameworks which often contain a context model, an event model and a sensor model, nicely fitting into our framework. In [96] we are exploring links between the presented approach and cognitive memory functions.

In general, we assumed a unique representation for an entity, for example we assume that $\mathbf{a}_{e_s}$ is the same in the prediction model and the semantic model. Sometimes it makes sense to relax that assumption and only assume some form of a coupling. [49, 7, 6] contain extensive discussions on the transfer of latent representations.

Appendix: Cost Functions

We consider cost functions for the KG tensor, the event tensor and the prediction model. The tilde notation $\underline{\tilde{X}}$ indicates subsets which correspond to the facts known in training. If only positive facts with $Value = True$ are known, as often the case in KGs, negative facts can be generated using, e.g., local closed world assumptions. We use negative log-likelihood cost terms. For a Bernoulli likelihood, $-\log P(x|\theta) = \log[1+\exp\{(1-2x)\theta\}]$ (cross-entropy) and for a Gaussian likelihood $-\log P(x|\theta) = const + \frac{1}{2\sigma^2}(x-\theta)^2$. We use regularization as described in Equation 8.2.

We describe the cost function in terms of the latent representations A and the M-mappings. W stands for the parameters in the functional mapping.

KG

The cost term for the semantic KG model is

$$\text{cost}^{KG} = -\sum_{x_{s,p,o} \in \underline{\tilde{X}}} \log P(x_{s,p,o}|\theta^{KG}_{s,p,o}(A,M,W))$$

Events

$$\text{cost}^{event} = -\sum_{z_{s,p,o,t} \in \underline{\tilde{Z}}} \log P(z_{s,p,o,t}|\theta^{event}_{s,p,o,t}(A,M,W))$$

Prediction Model

The cost function for the prediction model is

$$\text{cost}^{predict} = - \sum_{z_{s,p,o,t} \in \underline{\tilde{Z}}} \log P(z_{s,p,o,t} | \theta^{predict}_{s,p,o,t}(A, M, W))$$

Chapter 9

Conclusion

Along this thesis, we have shown multiple applications of Neural Networks with the goal of providing predictions and recommendations tailored to the specific profile and history of each patient. We have put special attention to one of the main challenges that araised when pursuing our goal, which is the ability to merge multiple source of static and dynamic information, and to be able to find complex patterns among all those variables. To tackle these challenges, we developed new Neural Network architectures and preprocessing pipelines to exploit the main characteristics of medical datasets.

In Chapter 2 we presented our Temporal Embeddings Model which shows how to integrate both temporal and medical features by using a Feedforward Neural Network. This model provides an acceptable performance for the task of modelling the next laboratory analysis and medication prescriptions, outperforming the other models that we tried. We showed how this Neural Network can perform intra-day recommendations (i.e. given the laboratory results obtained in the morning for a given patient, what medications should be prescribed in the afternoon), as well as full-visit recommendations (i.e. for any given patient, what laboratory tests should be made in the next visit, what are the expected results of such analysis and what medications should be prescribed next).

Interestingly, we found that the discretization of continuous variables into "high", "normal" and "low" categories, improved the performance of the model. We hypothesized that such improvement is a result of reducing the noise caused by imputation strategies and an increase of the parameters of the network.

Another interesting piece of information presented in Chapter 4 is the sensitivity analysis. It shows how our model learnt to predict the prescription of Tacrolimus, one of the medications that appears more frequently in the dataset, giving a very high importance to

the same observations that the physicians use to decide if this drug should be prescribed.

In Chapter 4, we show how an architecture based in Recurrent Neural Networks, which also integrated static information from the patients, outperforms the Temporal Embeddings Model in the task of predicting a set of endpoints after a kidney transplantation. In this chapter we predict the probability of observing a rejection, a graft loss or the death of a patient, both 6 and 12 month after the transplantation. The results obtained seem good enough to be useful in real life applications.

There is still room to improve the performance of the models in all the use cases presented in this thesis, specially in terms of false positive rate, which is usually the hardest aspect to improve in highly imbalanced datasets. Chapter 5 shows how some of these methods can be generalized to other use cases outside of the medical sector.

As stated before, this thesis pursues a very pragmatical goal, which is bringing Machine Learning technology into the daily clinical practice. For this reason, we also explored two fundamental aspects related with this purpose: how to integrate the models into a clinic or hospital, and how to create fully autonomous decisions systems.

Concerning the integration of ML models in hospitals, we designed and implemented a software architecture to enable physicians to interact with the developed models. Chapter 6 shows how we deployed the endpoint predictor at the Charité hospital in Berlin. We present the architecture the system designed for this purpose, as well as the software stack that we used.

Regarding the fully autonomous decision making model, in Chapter 7 we showed how Generative Adversarial Networks can be used to fully automate the process of drug prescriptions. This model converts the recommendations provided by a Neural Network into valid medical decisions. We showed with simulated data the benefits and limitations of this approach. The main downside of the method presented is that, in order to train such model, one would require a training dataset that includes all the valid prescriptions that a doctor could have made for each data sample.

Nevertheless, we consider that this kind of fully automated decision system could be highly useful due to the population ageing observed in many countries. The so called “silver tsunami” can translate into a saturation of the healthcare systems, and fully autonomous digital systems could be the solution for creating scalable healthcare services that solve this problem. This kind of system could for example start taking care of chronic patients suffering from certain diseases, where the spectrum of decisions that need to be taken could be pretty narrow. For this purpose, we proposed a system based on Generative Adversarial

Networks which is capable of modelling the behavior of an hypothetical human decision maker.

We live in a time in which the "big data" we generate is constantly exploited for commercial purposes. However, healthcare information systems have fallen behind. Very frequently the only exploitation of our healthcare records is made by physicians, who use their intuition, know-how and simple statistical models to do so. This is obviously an unacceptable situation and it is beyond any doubt that in the near future, physicians will be, at least, assisted by electronic systems that automatically process big amounts of clinical data. With our work we took the first steps towards this goal, and we believe that, with the effort of scientists and engineers, these systems will soon have a huge positive impact in our society.

Bibliography

[1] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 472:473, 2016.

[2] Emily Anthes. Mental health: There's an app for that. *Nature*, 532(7597):20–23, April 2016.

[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[4] Enrique Baca-Garcia, Maria de las Mercedes Perez-Rodriguez, Ignacio Basurte-Villamor, Jeronimo Saiz-Ruiz, José M Leiva-Murillo, Mario de Prado-Cumplido, Ricardo Santiago-Mozos, Antonio Artés-Rodríguez, and Jose de Leon. Using data mining to explore complex clinical decisions: A study of hospitalization after a suicide attempt. *The Journal of clinical psychiatry*, 67(7):1124–1132, July 2006.

[5] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

[6] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 17–36, 2012.

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[8] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *journal of machine learning research*, 3(Feb):1137–1155, 2003.

[9] Yoshua Bengio, Paolo Frasconi, and Patrice Simard. The problem of learning long-term dependencies in recurrent networks. In *Neural Networks, 1993., IEEE International Conference on*, pages 1183–1188. IEEE, 1993.

[10] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[11] Bettina Berendt, Andreas Hotho, and Gerd Stumme. Towards semantic web mining. In *International Semantic Web Conference*, pages 264–278. Springer, 2002.

[12] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

[13] Tim Berners-Lee. Linked data. design issues for the world wide web. *World Wide Web Consortium. http://www. w3. org/DesignIssues/LinkedData. html*, 2006.

[14] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[15] John J Bissler, J Christopher Kingswood, Elżbieta Radzikowska, Bernard A Zonnenberg, Michael Frost, Elena Belousova, Matthias Sauter, Norio Nonomura, Susanne Brakemeier, Petrus J de Vries, et al. Everolimus for angiomyolipoma associated with tuberous sclerosis complex or sporadic lymphangioleiomyomatosis (exist-2): a multicentre, randomised, double-blind, placebo-controlled trial. *The Lancet*, 381(9869):817–824, 2013.

[16] Hilario Blasco-Fontecilla, AnaLucia A Alegria, Jorge Lopez-Castroman, Teresa Legido-Gil, Jeronimo Saiz-Ruiz, Jose de Leon, and Enrique Baca-Garcia. Short

self-reported sleep duration and suicidal behavior: a cross-sectional study. *Journal of affective disorders*, 133(1-2):239–246, September 2011.

[17] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM, 2008.

[18] Antoine Bordes, Jason Weston, Ronan Collobert, Yoshua Bengio, et al. Learning structured embeddings of knowledge bases. In *AAAI*, volume 6, page 6, 2011.

[19] K Budde, F Lehner, C Sommerer, W Arns, P Reinke, U Eisenberger, RP Wüthrich, S Scheidl, C May, E-M Paulus, et al. Conversion from cyclosporine to everolimus at 4.5 months posttransplant: 3-year results from the randomized zeus study. *American Journal of Transplantation*, 12(6):1528–1540, 2012.

[20] Klemens Budde, Thomas Becker, Wolfgang Arns, Claudia Sommerer, Petra Reinke, Ute Eisenberger, Stefan Kramer, Wolfgang Fischer, Harald Gschaidmeier, Frank Pietruck, et al. Everolimus-based, calcineurin-inhibitor-free regimen in recipients of de-novo kidney transplants: an open-label, randomised, controlled trial. *The Lancet*, 377(9768):837–847, 2011.

[21] Kyunghyun Cho, Aaron Courville, and Yoshua Bengio. Describing multimedia content using attention-based encoder-decoder networks. *Multimedia, IEEE Transactions on*, 17(11):1875–1886, 2015.

[22] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[23] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[24] 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.

[25] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[26] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[27] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[28] Josef Coresh, Elizabeth Selvin, Lesley A Stevens, Jane Manzi, John W Kusek, Paul Eggers, Frederick Van Lente, and Andrew S Levey. Prevalence of chronic kidney disease in the united states. *Jama*, 298(17):2038–2047, 2007.

[29] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610. ACM, 2014.

[30] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[31] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[32] Daniel M Dunlavy, Tamara G Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(2):10, 2011.

[33] Cristóbal Esteban, Danilo Schmidt, Denis Krompaß, and Volker Tresp. Predicting sequences of clinical events by using a personalized temporal latent embedding model. In *Healthcare Informatics (ICHI), 2015 International Conference on*, pages 130–139. IEEE, 2015.

[34] Cristóbal Esteban, Danilo Schmidt, Denis Krompaß, and Volker Tresp. Predicting sequences of clinical events by using a personalized temporal latent embedding model. In *Healthcare Informatics (ICHI), 2015 International Conference on*, pages 130–139. IEEE, 2015.

[35] Nicola Fanizzi, Claudia dAmato, and Floriana Esposito. Dl-foil concept learning in description logics. In *International Conference on Inductive Logic Programming*, pages 107–121. Springer, 2008.

[36] Clemens Fartacek, Günter Schiepek, Sabine Kunrath, Reinhold Fartacek, and Martin Plöderl. Real-Time Monitoring of Non-linear Suicidal Dynamics: Methodology and a Demonstrative Case Report. *Frontiers in psychology*, 7(174102):130, 2016.

[37] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Aistats*, volume 15, page 275, 2011.

[38] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[39] Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE, 2013.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[41] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[42] Lu Huber, Marcel Naik, Klemens Budde, et al. Desensitization of hla-incompatible kidney recipients. *N Engl J Med*, 365:1643, 2011.

[43] Mathilde Husky, Emilie Olié, Sébastien Guillaume, Catherine Genty, Joel Swendsen, and Philippe Courtet. Feasibility and validity of ecological momentary assessment in the investigation of suicide risk. *Psychiatry research*, 220(1-2):564–570, December 2014.

[44] Eric Jones, Travis Oliphant, and Pearu Peterson. {SciPy}: open source scientific tools for {Python}. 2014.

[45] Ronald C Kessler, Christopher H Warner, Christopher Ivany, Maria V Petukhova, Sherri Rose, Evelyn J Bromet, Millard Brown, Tianxi Cai, Lisa J Colpe, Kenneth L Cox, et al. Predicting suicides after psychiatric hospitalization in us army soldiers: the army study to assess risk and resilience in servicemembers (army starrs). *JAMA psychiatry*, 72(1):49–57, 2015.

[46] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[47] Yehuda Koren, Robert Bell, Chris Volinsky, et al. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[48] Denis Krompaß, Cristóbal Esteban, Volker Tresp, Martin Sedlmayr, and Thomas Ganslandt. Exploiting latent embeddings of nominal clinical data for predicting hospital readmission. *KI-Künstliche Intelligenz*, 29(2):153–159, 2015.

[49] Hugo Larochelle, Dumitru Erhan, and Yoshua Bengio. Zero-data learning of new tasks. In *AAAI*, volume 1, page 3, 2008.

[50] Mark E Larsen, Tjeerd W Boonstra, Philip J Batterham, Bridianne O'Dea, Cecile Paris, and Helen Christensen. We feel: mapping emotion on Twitter. *IEEE J. of Biomedical and Health Informatics*, 19(4):1246–1252, July 2015.

[51] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, pages 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

[52] Jens Lehmann. Dl-learner: learning concepts in description logics. *Journal of Machine Learning Research*, 10(Nov):2639–2642, 2009.

[53] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.

[54] Gabriela Lindemann, Danilo Schmidt, Thomas Schrader, and Dietmar Keune. The resource description framework (rdf) as a modern structure for medical data. *International Journal of Biological and Medical Sciences*, 4(2), 2009.

[55] Matthias W Lorenz, Hugh S Markus, Michiel L Bots, Maria Rosvall, and Matthias Sitzer. Prediction of clinical cardiovascular events with carotid intima-media thickness a systematic review and meta-analysis. *Circulation*, 115(4):459–467, 2007.

[56] Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*, 2015.

[57] Michael J Lysaght. Maintenance dialysis population dynamics: current trends and long-term implications. *Journal of the American Society of Nephrology*, 13(suppl 1):S37–S40, 2002.

[58] Alexander Maedche and Steffen Staab. Ontology learning for the semantic web. *IEEE Intelligent systems*, 16(2):72–79, 2001.

[59] Wes McKinney. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, pages 1–9, 2011.

[60] T Mikolov and J Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 2013.

[61] Tomáš Mikolov. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 2012.

[62] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[63] Morten Mørup. Applications of tensor (multiway array) factorizations and decompositions in data mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):24–40, 2011.

[64] Travis B Murdoch and Allan S Detsky. The inevitable application of big data to health care. *Jama*, 309(13):1351–1352, 2013.

[65] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *arXiv preprint arXiv:1503.00759*, 2015.

[66] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

[67] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, volume 11, pages 809–816, 2011.

[68] Office of the Surgeon General (US) and National Action Alliance for Suicide Prevention (US). 2012 National Strategy for Suicide Prevention: Goals and Objectives for Action: A Report of the U.S. Surgeon General and of the National Action Alliance for Suicide Prevention. Technical report, US Department of Health & Human Services (US), Washington (DC), September 2012.

[69] Joshua O'Madadhain, Jon Hutchins, and Padhraic Smyth. Prediction and ranking algorithms for event-based network data. *ACM SIGKDD explorations newsletter*, 7(2):23–30, 2005.

[70] Deutsche Stiftung Organtransplantation. Nierentransplantation. *http://www.dso.de/organspende-und-transplantation/transplantation/ nieren-transplantation.html*.

[71] Daniel H OLeary and Joseph F Polak. Intima-media thickness: a tool for atherosclerosis imaging and event prediction. *The American journal of cardiology*, 90(10):L18–L21, 2002.

[72] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

[73] Heiko Paulheim, Petar Ristoski, Evgeny Mitichkin, and Christian Bizer. Data mining with background knowledge from the web. *RapidMiner World*, pages 1–14, 2014.

[74] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[75] Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[76] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 811–820. ACM, 2010.

[77] Achim Rettinger, Uta Lösch, Volker Tresp, Claudia d'Amato, and Nicola Fanizzi. Mining the semantic web. *Data Mining and Knowledge Discovery*, 24(3):613–662, 2012.

[78] Ryan A Rossi, Brian Gallagher, Jennifer Neville, and Keith Henderson. Modeling dynamic behavior in large evolving graphs. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 667–676. ACM, 2013.

[79] Francisco Ruiz, Isabel Valera, Carlos Blanco, and Fernando Pérez-Cruz. Bayesian Nonparametric Modeling of Suicide Attempts. In *Neural Information Processing Systems*, pages 1853–1861, 2012.

[80] Charles Safran, Meryl Bloomrosen, W Edward Hammond, Steven Labkoff, Suzanne Markel-Fox, Paul C Tang, and Don E Detmer. Toward a national framework for the secondary use of health data: an american medical informatics association white paper. *Journal of the American Medical Informatics Association*, 14(1):1–9, 2007.

[81] Sebastian Schneeweiss. Learning from big health care data. *New England Journal of Medicine*, 370(23):2161–2163, 2014.

[82] Kay Schröter, Gabriela Lindemann-v Trzebiatowski, and Lutz Fritsche. Tbase2a web–based electronic patient record. *Fundamenta Informaticae*, 43(1-4):343–353, 2000.

[83] Saul Shiffman, Arthur A Stone, and Michael R Hufford. Ecological momentary assessment. *Annual review of clinical psychology*, 4(1):1–32, April 2008.

[84] Amit Singhal. Introducing the knowledge graph: things, not strings, may 2012. *URL http://googleblog. blogspot. ie/2012/05/introducing-knowledgegraph-things-not. html*, 2012.

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

[86] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[87] Arthur A Stone. *The science of real-time data capture : self-reports in health research.* Oxford ; New York : Oxford University Press, 2007.

[88] Arthur A Stone and Saul Shiffman. Ecological momentary assessment (ema) in behavorial medicine. *Annals of Behavioral Medicine*, 1994.

[89] Arve Strandheim, Ottar Bjerkeset, David Gunnell, Sigrid Bjørnelv, Turid Lingaas Holmen, and Niels Bentzen. Risk factors for suicidal thoughts in adolescence–a prospective cohort study: the Young-HUNT study. *BMJ open*, 4(8):e005867–e005867, 2014.

[90] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.

[91] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 687–696. ACM, 2007.

[92] Jaana T Suokas, Jaana M Suvisaari, Marjut Grainger, Anu Raevuori, Mika Gissler, and Jari Haukka. Suicide attempts and mortality in eating disorders: a follow-up study of eating disorder patients. *General hospital psychiatry*, 36(3):355–357, May 2014.

[93] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[94] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012.

[95] John Torous, Patrick Staples, and Jukka-Pekka Onnela. Realizing the Potential of Mobile Mental Health: New Methods for New Data in Psychiatry. *Current Psychiatry Reports*, 17(8):61, June 2015.

[96] Volker Tresp, Cristóbal Esteban, Yinchong Yang, Stephan Baier, and Denis Krompaß. Learning with memory embeddings. *arXiv preprint arXiv:1511.07972*, 2015.

[97] Volker Tresp, Sonja Zillner, Maria J Costa, Yi Huang, Alexander Cavallaro, Peter A Fasching, André Reis, Martin Sedlmayr, Thomas Ganslandt, Klemens Budde, et al. Towards a new science of a clinical data intelligence. *arXiv preprint arXiv:1311.4180*, 2013.

[98] Mickey Trockel, Bradley E Karlin, C Barr Taylor, Gregory K Brown, and Rachel Manber. Effects of cognitive behavioral therapy for insomnia on suicidal ideation in veterans. *Sleep*, 38(2):259–265, January 2015.

[99] Julia E Vogt, Marius Kloft, Stefan Stark, Sudhir S Raman, Sandhya Prabhakaran, Volker Roth, and Gunnar Rätsch. Probabilistic clustering of time-evolving distance data. *Machine learning*, 100(2-3):635–654, 2015.

[100] Fei Wang, Noah Lee, Jianying Hu, Jimeng Sun, and Shahram Ebadollahi. Towards heterogeneous temporal clinical event pattern discovery: a convolutional approach. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 453–461. ACM, 2012.

[101] World Health Organization (WHO). Preventing suicide: A global imperative, 2014.

[102] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, and Jaime G Carbonell. Temporal collaborative filtering with bayesian probabilistic tensor factorization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 211–222. SIAM, 2010.

[103] Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.

[104] MWB Zhang, CSH Ho, CCS Cheok, and RCM Ho. Smartphone apps in mental healthcare: the state of the art and potential developments. *BJPsych Advances*, 2015.