COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS PHYSICS AND INFORMATICS

PREDICTION OF HEALTH STATUS DETERIORATION

Master thesis

2025 Bc. Marián Kravec

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS PHYSICS AND INFORMATICS

PREDICTION OF HEALTH STATUS DETERIORATION

Master thesis

Study program:	Applied informatics
Branch of study:	Applied informatics
Department:	Department of Applied Informatics
Supervisor:	MSc. František Dráček
Consultant:	

Bratislava, 2025

Bc. Marián Kravec

58379976

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Marián Kravec
Študijný program:	aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Prediction of Health Status Deterioration
Predikcia zhoršenia zdravotného stavu

Anotácia: V súčastnosti sa sektor zdravotníctva na Slovensku vyznačuje nizkou mierou využita dostupnych zdravotníckych dát. V rámci tejto prace je cieľom ukázať, že z existujúcich dát je možné predikovať vyvoj dalšieho zdravotného stavu pacienta, poprípade odhadnúť vývoj budúcich nákladov za účelom lepšieho plánovania prerozdelenia financí v rámci sektoru.

Cieľ: Práca bude rozdelená na dve časti, v prvej študent urobí teoretické zhrnutie existujúchich metód spracovania dát a metód strojového učenia, ktoré sa budú dať potenciálne aplikovať na daný problém. V druhej časti navrhne a aplikuje predičkný model.

Literatúra: T. Sk, L. M. G, L. R. K and R. R. J, "Health Status Prediction using ML Techniques," 2022 6th International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2022, pp. 1191-1196, doi: 10.1109/ICCMC53470.2022.9753766.

Jödicke, A.M., Zellweger, U., Tomka, I.T. et al. Prediction of health care expenditure increase: how does pharmacotherapy contribute?. BMC Health Serv Res 19, 953 (2019). https://doi.org/10.1186/s12913-019-4616-x

Vedúci:	MSc. František Dráček
Katedra:	FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry:	doc. RNDr. Tatiana Jajcayová, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 05.10.2023

Dátum schválenia: 05.10.2023 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

[illegible]

študent vedúci práce

I hereby declare that I have written this thesis by myself, only with help of referenced literature, under the careful supervision of my thesis advisor.

. . . .

Bratislava, 2025 Bc. Marián Kravec

Acknowledgment

I would like to express my sincere gratitude to all those who supported me throughout the journey of completing this diploma thesis.

First and foremost, I am deeply thankful to my supervisor, František Dráček, for his invaluable guidance, encouragement, and insightful feedback during every stage of my research. His expertise and patience have been instrumental in shaping this work.

I would also like to thank the Faculty of Mathematics, Physics and Informatics at Comenius University for providing an inspiring academic environment and access to state-of-the-art research facilities. The faculty's commitment to excellence in mathematics, physics, and informatics has greatly enriched my academic experience.

Special thanks go to my colleagues and friends for their support, stimulating discussions, and for making this journey memorable.

Finally, I am profoundly grateful to my family for their unwavering love, patience, and motivation. Their belief in me has been my greatest source of strength.

Abstract

Currently, the healthcare sector in Slovakia is characterized by a low rate of utilization of available healthcare data. The aim of this work is to show that from existing data it is possible to predict the development of the patient's further health status, or to estimate the development of future costs in order to better plan the redistribution of finances within the sector.

Keywords: cost prediction, patient future, transformers

Abstrakt

V súčastnosti sa sektor zdravotníctva na Slovensku vyznačuje nízkou mierou využitia dostupných zdravotníckych dát. V rámci tejto prace je cieľom ukázať, že z existujúcich dát je možné predikovať vývoj ďalšieho zdravotného stavu pacienta, poprípade odhadnúť vývoj budúcich nákladov za účelom lepšieho plánovania prerozdelenia financií v rámci sektoru.

Kľúčové slová: predikcia ceny, budúcnosť pacienta, transformery

Contents

List of Figures

List of Tables

Terminology

Terms

Abbreviations

- **CPT** - Current Procedural Terminology.
- **EHR** - Electronic Health Records.
- **ICD-10-CM** - International Classification of Diseases, Tenth Revision, Clinical Modification.
- **MKCH-10** - International Classification of Diseases, Tenth Revision (Medzinárodná klasifikácia chorôb).
- **ATC** - Anatomical Therapeutic Chemical.
- **LLM** - Large language model.
- **LaBSE** - Language-agnostic BERT sentence embedding model.
- **BERT** - Bidirectional encoder representations from transformers.
- **KNN** - K-nearest neighbors algorithm.
- **FFNN** - Feed forward neural network.
- **MLP** - Mutlilayer perceptron.
- **MLM** - Masked Language Model.
- **NSP** - Next Sentence Prediction.
- **TLM** - Translation Language Model.
- **PCA** - Principal component analysis.
- **RNN** - Recurrent neural network
- **SRN** - Simple recurrent network
- **GRU** - Gated recurrent unit RNN
- **LSTM** - Long short-term memory RNN

Motivation

State governments and insurance companies collect and store vast amounts of medical data, including patient histories, treatment records, prescriptions, and billing information. This data can be used for various purposes, such as detecting fraudulent activities, tracking contagious disease outbreaks, or allocating future supplies of purchased medication.

Another interesting application is predicting a patient's future health, which allows for forecasts of potential diseases they may develop and the treatments they might require. Unfortunately, due to the large number of significant factors not captured in medical records, this task is almost impossible to accomplish accurately.

That's why, in this study, we focus on a simpler problem: predicting the expected cost for a patient in the next year. Specifically, we aim to estimate the total anticipated costs of medications and medical procedures provided to a patient over a one-year period. In general, our approach involves two main tasks. First, we embed patient records into numerical vectors. Second, we train a model to predict each patient's future costs based on their previous records.

In the very first chapter, we briefly introduce our goal, the challenges we encountered, and the methods we used. The second chapter provides a quick overview of studies that address similar problems. The third chapter is dedicated to introducing the data we used. In the fourth chapter, we present the models and techniques employed for embedding records and for the prediction task. The fifth chapter is a brief section on the technical aspects of software design, including the programming languages and libraries we used. After that, in the sixth chapter, we describe how we implemented solutions to our two main tasks. The seventh chapter focuses on preliminary research, specifically the testing of embedding methods and various model parameters. Finally, in the eighth chapter, we discuss the results of the final model.

Chapter 1

Introduction

In this chapter, we briefly introduce our goal, the challenges we encountered, and the methods we used.

The main goal of our study is to develop software capable of analyzing patients' historical records-specifically, the medications prescribed to them and the medical procedures they have undergone-in order to predict the cost category each patient will belong to in the following year. In other words, we aim to estimate the expected expenses for each patient over the next year based on their prior medical data.

To achieve the desired outcome, we faced three primary challenges. First, we needed to transform patients' historical records into numerical vectors suitable as input for machine learning models. Second, we had to simulate patients' potential futures by predicting both likely medication prescriptions and medical procedures they might undergo in the subsequent year. Finally, we required a method to calculate the expected costs of these projected treatments and aggregate them into an annual cost estimate for each patient.

For record transformation (referred to as embedding later in the study), we focused on embedding medications, diagnoses, and medical procedures. We employed one of two methods depending on whether the data contained structured codes. When structured codes were available, we split the code into its hierarchical components, embedded each part individually, and then combined the results to create a comprehensive representation. For data without structured codes or meaningful organization, we utilized a machine learning-based language model to generate embeddings directly from textual descriptions.

To generate future medical records, we evaluated several sequential data models, including LSTM (Long Short-Term Memory) networks and decoder-only Transformer architectures, as patients' future medications and procedures exhibit strong temporal dependencies on their historical records, with recent events carrying greater predictive weight.

To estimate the cost category of each medical record, we primarily utilized a Multi-layer Perceptron (MLP) model. This neural network takes the numerical representation of a record (generated through our embedding process) as input and predicts its likely cost category. While the MLP served as our core approach, we also explored alternative models such as Gradient Boosting and Ridge Regression for comparative analysis.

After resolving the three key challenges we integrated these components into a unified software solution. This final system sequentially processes patient histories through each stage to generate annual expense forecasts.

Chapter 2

Similar studies

A necessary prerequisite for predicting a patient's future is embedding patient records into numerical representations. This process is relatively straightforward for attributes that are already numeric, such as age, or for attributes with structured codes, like diagnoses. However, with medical procedures, a challenge arises: the codes associated with procedures lack a hierarchical structure. As a result, two similar procedures might have completely different codes. To address this, we needed a method to embed procedures so that similar procedures receive similar embeddings. One effective approach is to group medical procedures into clusters, assigning similar embeddings to procedures within the same cluster and dissimilar embeddings to those in different clusters.

Lorenzi et al. from Duke University in Durham developed a novel algorithm called Predictive Hierarchical Clustering [25]. This algorithm was designed for agglomerative clustering of surgical CPT codes. It uses a one-pass, bottom-up approach that utilizes EHR data-specifically, 317 predictors such as lab values and patient history, while excluding CPT information-from 3 723 252 patients and 3 132 CPT codes, where each patient has one main surgical CPT code. For each CPT code, they create a tree containing patients with the corresponding code. Afterwards, at each iteration, the algorithm considers merging all pairs of existing trees. To compare two trees, it uses two hypotheses: the first hypothesis states that the data in both trees are generated from the same model, while the second states that the data in each tree are generated from models with different parameters. The final value is a weighted average of the probabilities of these two hypotheses, considering the data in the trees, where the weight is the probability of the first hypothesis which formula is shown in Eq. 2.1.

$$p(D_k|T_k) = p(H_1^k)p(D_k|H_1^k) + (1 - p(H_1^k))p(D_i|T_i)p(D_j|T_j) \tag{2.1}$$

Where D_k is the set of data in the merged tree (formed by merging T_i and T_j), T_k is the merged tree, H_1^k is the first hypothesis, and D_i and D_j are the data in trees

T_i and T_j. During experimental testing, they compared the results of their approach to clustering CPT codes into 16 clinical groups. To evaluate performance, they used both their clustering method and clinical clustering to predict additional procedures for patients in a validation group, then computed the area under the receiver operating characteristic curve (AUROC) and the area under the precision-recall curve (AUROC). Their model achieved a few percentage points of improvement in these metrics compared to clinical clustering.

In the end, we decided not to use this approach in our work, as it cannot reliably distinguish whether two procedures are considered similar due to their actual similarity or simply because they frequently occur together as part of the determination or treatment of a specific diagnosis. This distinction is important, since procedures that only appear together may have vastly different costs associated with them, which could pose issues for the prediction model.

The main goal of our study is to predict the future cost for each patient, specifically how much money will be spent on drugs and procedures for a given individual.

One similar study in this regard is by Caballer-Tarazona et al. [12], which aimed to predict patients' future healthcare costs using an "Aggregated Clinical Risk Group 3" (ACRG3) variable derived from standardized Clinical Risk Groups (CRGs). The ACRG3 variable consists of two components: the first assigns patients to one of nine grouped CRGs, and the second categorizes them into one of six severity levels. The authors also tested whether adding demographic variables like age or sex improved model performance.

To address the high proportion of zero-cost observations, they developed a two-part model:

1. First part: A logit model estimating the probability of incurring non-zero costs.
2. Second part: Either a log-linear ordinary least squares (OLS) regression or a generalized linear model (GLM) with a log link to predict expected costs for patients with positive expenditures.

The final predicted cost was calculated as the product of the probability from the first part and the conditional expectation from the second part. Their models achieved adjusted R^2 values of 46.4%–49.4%, which they noted as comparable to results from studies using alternative patient classification systems.

This approach was ultimately not viable for us, as it relies on specific attributes such as CRG and patient severity level, which we did not have available.

Another study focused on future patient cost prediction is by Mohammad Amin Morid et al. [29], which compares multiple models for predicting a patient's future cost category using medical and pharmacy claims data.

For each patient, their dataset included 21 features, all related to the amounts spent on that patient. Specifically, these features included values such as the total historical cost for the patient, the total cost in the last six months, the number of months with costs above the patient's average, and the linear trend of cost over time.

The authors aimed to predict into which of five cost categories a patient would fall in the future. To achieve this, they compared several predictive models, including Linear Regression, Decision Tree, and Artificial Neural Network (ANN). For Linear Regression, they also tested versions with regularization, such as Lasso and Ridge models. In the case of Decision Tree models, they evaluated improved ensemble methods based on decision trees, including Random Forest, Bagging, and Gradient Boosting techniques.

In their testing, the most successful model turned out to be Gradient Boosting, which was able to predict the correct cost category in almost 95% of cases. However, for the highest cost bucket-which was expected to contain only 2% of the population-the model was successful in just 37.5% of cases. The second and third best models were the ANN and Ridge models, both of which achieved very similar results, with around 91.5% overall accuracy in category assignment and over 50% correct classification in each cost category. In conclusion, they determined that Gradient Boosting generally provides the best performance for cost prediction models, while ANN offers superior results specifically for higher-cost patients.

This study gave us hope that our data might contain the necessary information for accurate prediction, as most of the features in their study were computed using information about the cost of drugs or procedures-data that is present for each record in our dataset, which also contains much more detail about the causes of those costs.

Chapter 3

Medical data

Our models were developed for a database of health insurance claims maintained by the Slovak National Health Information Center (NCZI). We focused primarily on datasets related to drug prescriptions and medical procedures administered by outpatient healthcare providers.

To train and verify the model, we used completely artificial data with a structure matching that of the NCZI database. The data were generated in a way that simulates realistic patient records.

The data we have can be split into two categories: patient records and a mapping table.

3.1 Patients records

Patients records are split into two dataset:

- Records of medical procedures from outpatient healthcare
- Records of prescribed drugs

In total, we have data on 173 355 patients. Each patient has at least 70 records, resulting in a dataset with 133 679 705 records overall.

3.1.1 Records of medical procedures from ambulatory health care

Each row of this dataset corresponds to a single patient record, specifically representing one medical procedure performed on that patient. Each record consists of the following variables:

- date of the procedure - date when procedure was performed
- code of the patient - identification code unique for the patient
- age of the patient - age of the patient at the time of procedure
- gender of the patient
- code of the diagnosis - identification code unique for the diagnosis for which procedure was prescribed
- code of the procedure - identification code unique for the medical procedure
- cost of the procedure - cost associated with performing of the procedure

For our prediction model, we use most of this information. The date of the procedure, combined with the patient's age, generates timestamp data used to chronologically order all records for a patient and serves as one dimension of the record embedding. The patient identification code enables the aggregation of all records for a single individual. Diagnosis and procedure codes are mapped to their corresponding numerical vectors and embedded into the record's vector representation (see 5.1). Finally, cost is encoded into a cost category and serves as the target variable for training the record cost prediction models.

3.1.2 Records of prescribed medicines

Similarly to the dataset containing procedures, each row in this dataset corresponds to a single patient record, in this case representing one drug prescription for a specific patient. Each record consists of the following variables:

- date of the prescription - date when drug was prescribed
- code of the patient - identification code unique for the patient
- age of the patient - age of the patient at the time of procedure
- gender of the patient
- code of the diagnosis - identification code unique for the diagnosis for which drug was prescribed
- code of the drug - identification code unique for the drug
- cost of the drug - cost associated with performing of the procedure

The use of these variables is also similar to that for procedures, with the only difference being that, instead of encoding the procedure into the record embedding, we encode the drug.

3.2 Mappings

In addition to the patient records dataset, we utilize three mapping files to translate identification codes from the raw patient records into their corresponding embedding vectors. These mapping datasets are:

- Diagnosis mapping
- Drug mapping
- Medical procedure mapping

These datasets are publicly available dimension tables maintained and used by NCZI. In general, all three mapping tables have a comparable structure consisting of three main attributes. The first is an internal identification number used by NCZI to join these mappings to patient records. The second attribute is the official code; in the case of drugs and diagnoses, this is an internationally standardized structured code, while medical procedures use a code specific to Slovakia, which unfortunately lacks any meaningful structure. The final important part is the textual description.

We used these dimension tables to generate embedding vectors for drugs, diagnoses, and medical procedures. The process of this embedding is explained in Sec. 7.1.

Chapter 4

Theory - models

During our research, we utilized several different machine learning models. In this chapter, we introduce these models from a more theoretical point of view.

4.1 Multilayer perceptron

A multilayer perceptron is a feed-forward neural network, meaning that data flows in a single direction and neurons do not form cycles. This network consists of fully connected, sometimes called dense, layers with non-linear activation functions, as shown in Fig. 4.1. We can see that this model can be split into three parts: the input layer, which loads the data; the hidden layers, which extract the desired information using linear transformations and activation functions; and finally, the output layer, which applies a final linear transformation followed by an activation function to produce the output. Each layer can be described using the formula shown in Eq. 4.1, where h_i is the resulting vector of the i-th layer, $act()$ is a non-linear activation function, W_i is the weight matrix of the i-th layer, h_{i-1} is the resulting vector from the previous layer, and b_i is the bias vector.

$$h_i = act(W_i h_{i-1} + b_i), \tag{4.1}$$

4.2 Recurrent neural network

In general, a recurrent neural network is a type of neural network that, in some way, uses results from previous steps to improve its predictions. These models are used for ordered data, where each subsequent step depends on more than just the immediately preceding one.

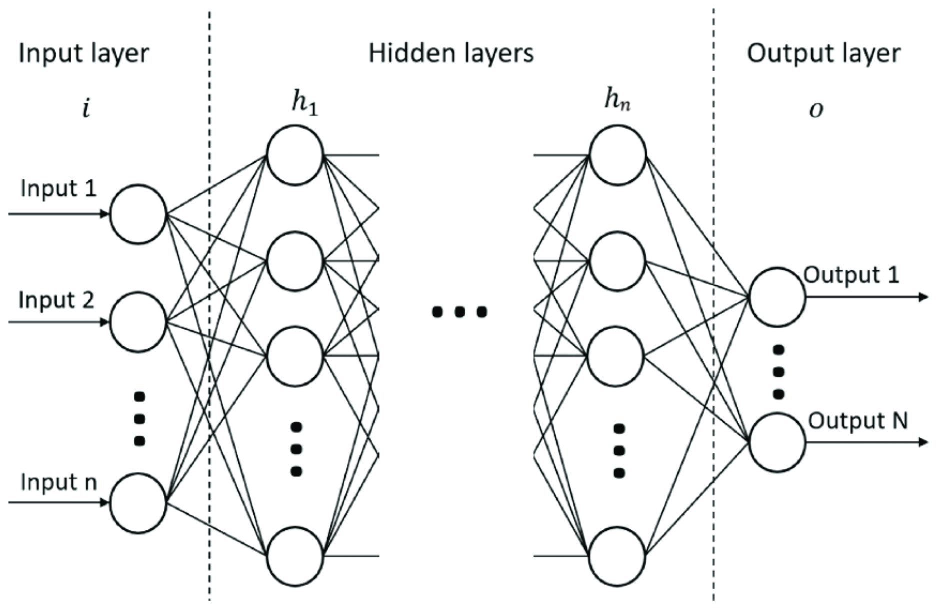


Figure 4.1: Architecture of multilayer perceptron [22].

4.2.1 Elman RNN

The Elman RNN, also known as a simple recurrent network (SRN), is a type of recurrent neural network that utilizes the results of the hidden layer before activation in the previous step as an additional input in the next one. We can see this architecture in Fig. 4.2, where on the left we observe how forward propagation occurs and on the right how it is unrolled over time. Here, the middle layer, which is the hidden layer of the model, receives two inputs: the input vector x_t, where t denotes the step (usually a time step), which is multiplied by a matrix of weights W_i, and the vector h_{t-1}, which is the result of this layer (also called the context vector) from the previous step, multiplied by a different matrix of weights denoted as W. These two results are summed together, and the result is passed to the activation layer, which generates the new vector h_t [19]. This entire process is summarized in Eq. 4.2, where b_i and b are bias vectors.

$$h_t = act(x_t W_i^T + b_i + h_{t-1} W^T + b). \tag{4.2}$$

Usually, this result is directly used as the output or passed through a single fully-connected feed-forward layer. In a multi-layered version of this network, the output of the first hidden layer is fed into the next hidden layer, along with the output of that specific layer from the previous step. Both inputs are multiplied by weight matrices specific to that layer, summed together, and then passed through an activation function.

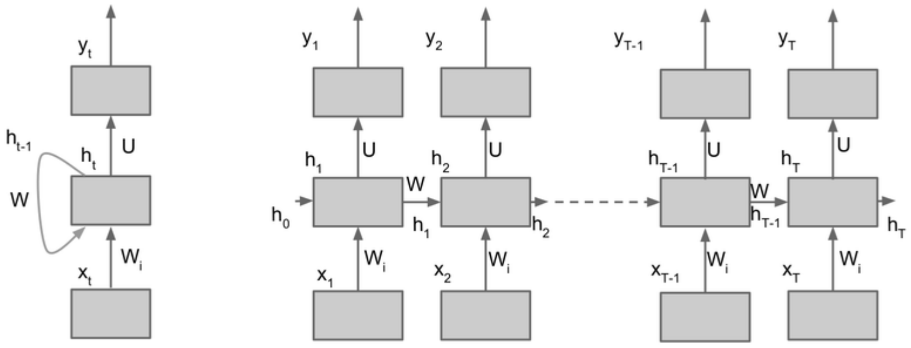


Figure 4.2: Architecture of Elman RNN [8].

4.2.2 Gated Recurrent Unit

A Gated Recurrent Unit (GRU) is a more complex RNN variant compared to the Elman RNN. Developed as a simplification of the even more intricate LSTM model, the GRU primarily consists of three interacting components: the reset gate, update gate, and candidate hidden state computation. These components collaboratively regulate information flow to produce the final prediction.

The structure of a GRU's hidden layer is illustrated in Fig. 4.3. In this diagram:

- Sigmoid: Represents a fully connected feed-forward layer with a sigmoid activation function.
- Tanh: Represents a fully connected feed-forward layer with a hyperbolic tangent activation function.

The reset gate, located on the left side of the diagram, uses the input and previous hidden state vectors to modify the previous hidden state vector, which is then fed into the candidate hidden state computation. This process helps capture short-term dependencies in time series by selectively removing information from the previous hidden state vector.

The candidate hidden state computation is similar to the Elman RNN but with two key differences: first, the inputted hidden layer is modified by the reset gate's result; second, the resulting vector from this part serves only as a candidate that undergoes further calculation. This candidate vector primarily contains current and short-term past information due to the specific vectors that form its input.

The update gate, similar to the reset gate, uses the concatenation of the input and previous hidden state vector as its input. However, its results are used to modify both the previous hidden state and the candidate hidden state, creating a final hidden state that is a weighted average of these two vectors, with weights derived from this gate. This architecture helps capture long-term dependencies in time series by reintroducing relevant information from the previous hidden state vector into the final hidden state, combining it with the candidate hidden state vector that contains mostly current and short-term information. The entire architecture is depicted in Fig. 4.3.

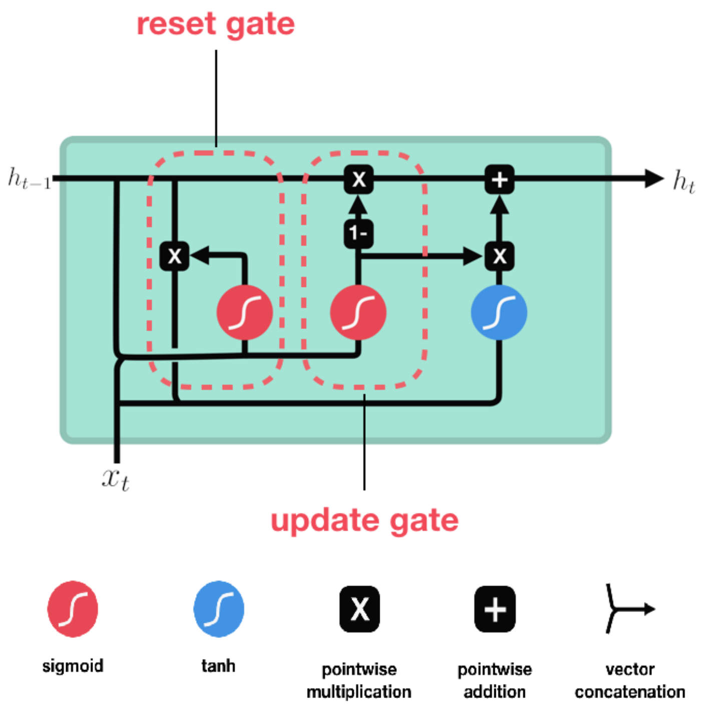


Figure 4.3: Architecture of GRU hidden layer.

The multi-layered version of this architecture is achieved by stacking hidden layers, where the hidden state vector of one layer becomes the input vector for the next.

4.2.3 Long Short-Term Memory

As mentioned earlier, the LSTM is a more complex predecessor of the GRU. Developed to address the vanishing gradient problem that limits long-term memory in models like the Elman RNN, the LSTM introduces a memory vector, often called a memory cell, designed to maintain information over longer periods. As shown in Fig. 4.4, this model consists of three gates and a candidate memory computation. The legend in this diagram has the same meaning as in the GRU architecture diagram.

All three gates and the candidate memory computation share the same input, which consists of the input vector and the previous hidden state vector. In each case, this input passes through a fully connected feed-forward layer and then into an activation function. For the gates, this activation function is the sigmoid function, which bounds all values within the range (0,1), while the candidate memory calculation uses the hyperbolic tangent as its activation function.

The forget gate is responsible for removing unnecessary information from the memory vector by performing an element-wise multiplication of its result with the previous memory vector.

The input gate, together with the candidate memory vector computation, is designed to introduce new information into the memory cell after adjustments made by the forget gate. First, the model computes the candidate memory vector; this vector is then adjusted by the input gate's result and added to the modified previous memory vector, resulting in a new memory vector.

Finally, the output gate determines how much each part of the memory should contribute to the new hidden state vector. This process is illustrated in Fig. 4.4.

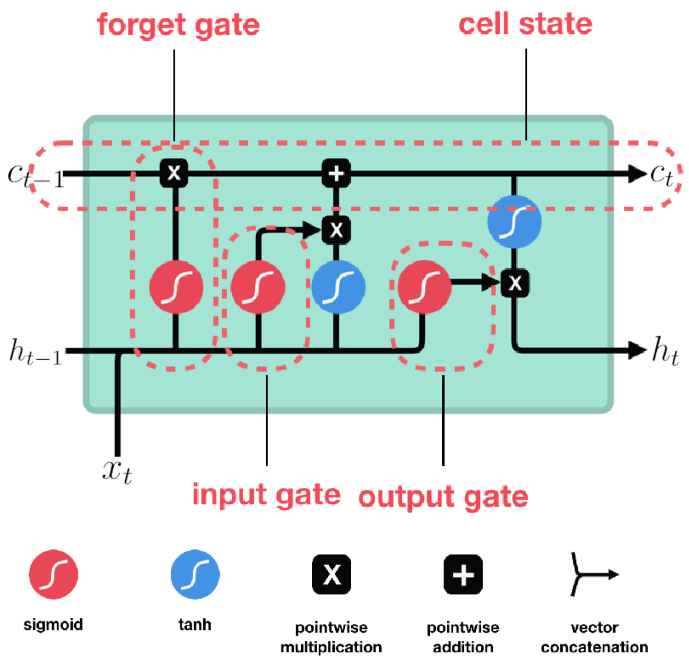


Figure 4.4: Architecture of LSTM hidden layer.

4.3 Transformer

The Transformer is a deep learning model architecture introduced in 2017 by Vaswani et al. [38] that is especially good at handling sequences, such as text. This architecture consists of an encoder and a decoder, as shown in Fig. 4.5. The encoder in this model is meant to create a contextualized representation of the input, while the decoder takes this contextualized representation and combines it with previous outputs to predict the next output. In addition to these two main blocks, this model usually also contains a tokenizer, an embedding layer, and positional encoding to prepare the input data, as well as a fully connected feed-forward layer with a softmax activation function to turn the result of the decoder into a probability distribution over possible tokens.

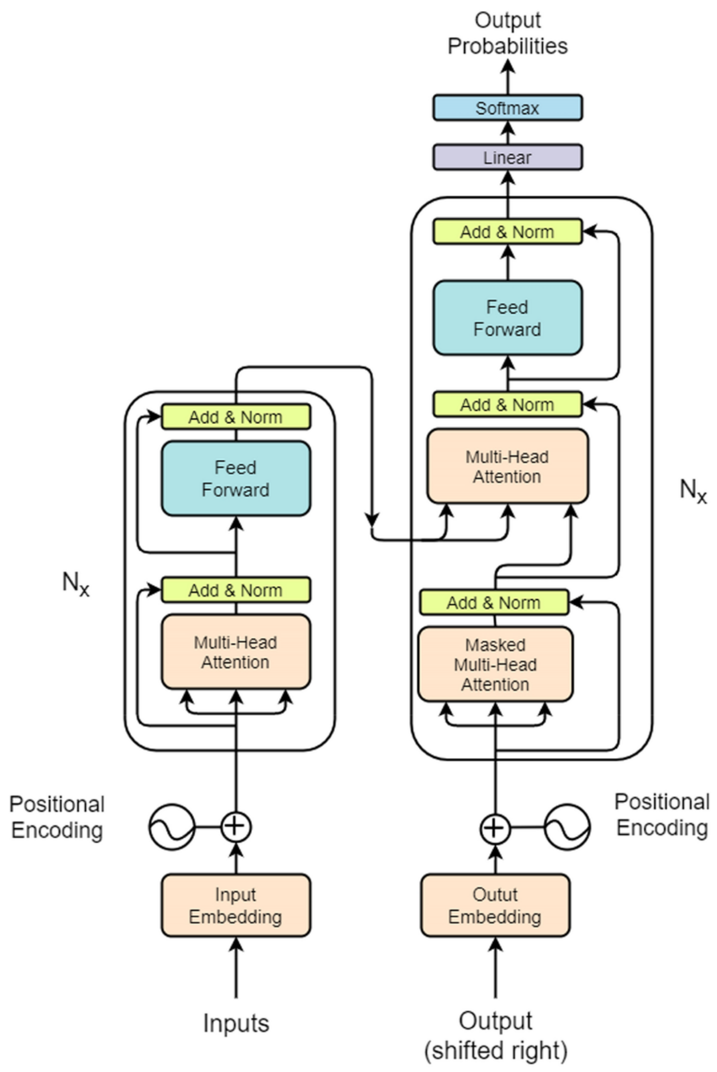


Figure 4.5: Architecture of one encoder-decoder block in transformer model from original "Attention is all you need" paper [38].

4.3.1 Tokenizer, Embedding layer and Positional encoding

The first part of the Transformer model is usually input preparation, which may vary depending on the type of data used. This part consists of three components:

- Tokenizer: This layer splits the input into tokens. For example, if the input is textual, tokens might be words or syllables.
- Embedding layer: The task of this layer is to transform each input token into a numerical vector of the same length.
- Positional encoding: The final preparation layer adds a vector to each token's embedding, encoding its position relative to other tokens.

In most cases, the Tokenizer and Embedding layer are trained separately and do not change during training of the Transformer, while the Positional Encoding is trained alongside the rest of the Transformer.

4.3.2 Encoder

Encoder architecture consists of multiple attention blocks, where each block consists of multi-headed self-attention and a multi-layered feed-forward neural network. After each of these steps, the embeddings from the step input are added to the output, and the result is layer-normalized. Adding the step input embedding creates residual paths that help mainly with the vanishing gradient problem, while layer normalization ensures that the results neither explode nor vanish, and also brings a bit of additional non-linearity to the model. First, input embeddings are split into parts, each of which goes into a separate self-attention head.

The architecture of self-attention is shown in Fig. 4.6, where we can see that each input token is multiplied by key, query, and value matrices to obtain their key, query, and value vectors. After that, each query is multiplied with each key to create the attention matrix, which encodes how much each token influences the others. This matrix then goes into a column-wise softmax function to normalize it, and finally, it is multiplied by the matrix of value vectors to create new embeddings of the tokens that should contain not only the original information but also the influence information.

The results of each self-attention head are then concatenated back into new embeddings that have the same dimensions as the original ones. After adding the residual connection and normalization, the results go into a multi-layered feed-forward neural network to further extract information from the embeddings.

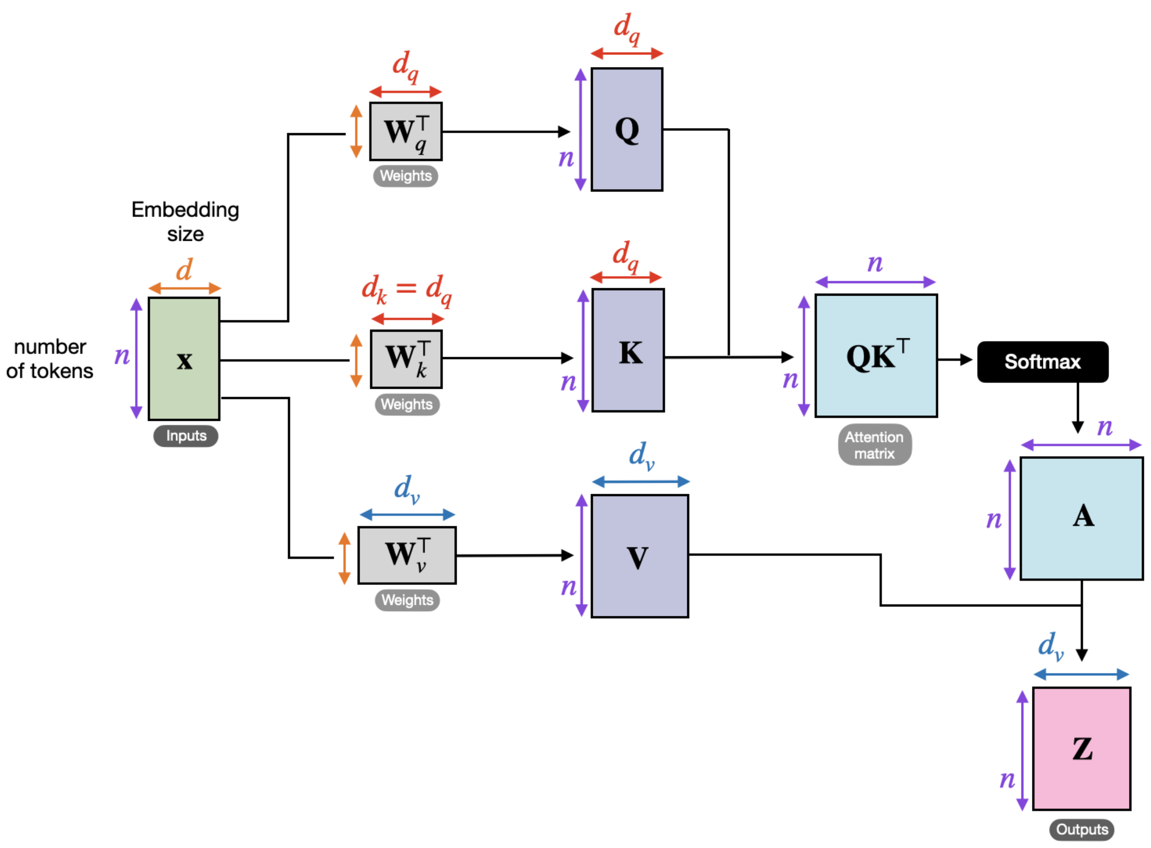


Figure 4.6: Architecture of Self-attention mechanism.

4.3.3 Decoder

The architecture of the decoder, similarly to the encoder, consists of multiple attention blocks; however, in this case, each attention block consists of three parts instead of two.

The first part is masked multi-headed self-attention, which is similar to unmasked self-attention, with the only difference being the application of masking to the attention matrix before the softmax function. This ensures that words at certain positions do not affect words at other specific positions. The decoder uses what is called causal or look-ahead masking, which prevents future tokens from affecting past ones. In our case, this guarantees that a record cannot be influenced by records that occur later, that is, in the future from its perspective.

The second part is multi-headed cross-attention, which takes two lists of token embeddings and computes the effect of tokens in the first list on tokens in the second list. In this case, the key and value vectors are computed from the contextualized representation produced by the encoder, while the query vectors are computed from the results of self-attention in the decoder. Other than this, the remaining architecture is the same as unmasked self-attention..

The final part of the decoder attention block is a multi-layered feed-forward neural network that further extracts information from the embeddings.

4.3.4 Fully-connected Feed-forward layer

After that, the results of the decoder pass into the final fully-connected feed-forward layer, which changes the dimensionality of the decoder output from the embedding size to the vocabulary size and applies a softmax activation to the result. This setup is designed to produce a probability distribution over all possible tokens for the last token.

In our case, we modified this final step. We encountered a problem where most records (our tokens) were unique, creating an extremely large vocabulary. This issue arose partially due to the many possible combinations of diseases, drugs, and medical procedures encoded in each record, and partially due to timestamps adding uniqueness, as it is unlikely for two patients to receive the same drug for the same disease at the same age.

To address this, we removed the softmax activation and altered the fully-connected feed-forward layer so that the result maintains the embedding dimension. We then added a function that splits the result, finds the closest embedding for each part, and concatenates these embeddings together, yielding a specific new embedding prediction instead of a probability distribution.

4.3.5 Decoder-only Transformer

The Decoder-only version of the Transformer model is a simplified variant that entirely excludes the Encoder part and removes the Cross-Attention mechanism from the Decoder. This model is typically used when generating subsequent tokens without relying on a stable contextual input that would normally be processed by the Encoder.

This simplification makes the architecture more akin to a standard RNN in terms of input-output behavior, where the model generates sequences step-by-step without explicit cross-contextual dependencies.

4.4 Language-agnostic BERT Sentence Embedding

The Language-agnostic BERT Sentence Embedding model, also known as LaBSE, is a model trained with the main goal of generating similar representations for pairs of sentences that have the same meaning and are only translations of each other in two different languages [4].

The architecture of the LaBSE model consists of four parts [7]:

1. Encoder-only transformer (BERT model)
2. Pooling layer
3. Dense layer
4. Normalization layer

4.4.1 Encoder-only Transformer (BERT model)

The first and most important part of the LaBSE model is the transformer, a deep learning architecture. More specifically, LaBSE uses BERT (Bidirectional Encoder Representations from Transformers), an encoder-only transformer architecture. This means the model lacks the decoder found in the standard Transformer (typically used for prediction tasks), allowing BERT to focus solely on extracting contextual information from input text.

The architecture of the standard BERT model includes:

1. Tokenizer layer
2. Embedding layer
3. Encoder
4. Task layer

Tokenizer layer

The first layer is the tokenizer, which takes the input text and splits it into tokens. In the case of the BERT model, this is called the PieceWise tokenizer, which splits text into subwords, something similar to syllables. The PieceWise tokenizer has advantages compared to other tokenizers that use either words or characters. Compared to character-wise tokenization, subwords contain more information than individual characters. Compared to word tokenization, there are far fewer subwords than words, and subwords are more similar across multiple languages, resulting in a much smaller vocabulary. This is especially important for multilingual models. After splitting, this layer assigns an integer number to each unique token. The LaBSE model's vocabulary distinguishes around 500,000 different tokens.

Embedding layer

After that comes the embedding layer, which assigns a real-number vector to each token. Specifically, the BERT model computes three distinct embeddings, sums them, and normalizes the result to produce the final embedding:

- Token type embedding: The base embedding where each token in the vocabulary is assigned a unique vector.
- Positional embedding: Encodes the token's position within the sequence, providing contextual information about its location.
- Segment type embedding: Indicates which segment (typically a sentence) the token belongs to, crucial for processing multi-sentence input.

Encoder

The third and most important layer is the encoder. This is the layer in which contextual information is mined from the text. The architecture of this layer is the same as the encoder described in Sec. 4.3.2. In the BERT variant used by LaBSE, the encoder contains 12 attention blocks.

Task layer

The training process of the BERT model usually consists of two tasks on which the model is trained at the same time. The first is Masked Language Modeling (MLM), where 15% of input tokens are masked, meaning they are either replaced by a mask placeholder or by a random different token. The masked input then goes into the model, and the resulting tokens at the positions of the masked tokens in the input are compared to the correct tokens before masking. This creates an error that is back-propagated through the model, updating its parameters [17]. The other task usually used is called Next Sentence Prediction (NSP). In this task, the model receives input that starts with a special classification token and then two spans of text separated by a special separator token. The task of the model is to determine whether these two spans of text can appear one after another, or more precisely, whether they appeared consecutively in the training corpus. This information is encoded in the first token of the result using two special tokens, either "is next" or "not next." Similarly, the difference between the expected and resulting first token creates an error that is back-propagated through the model [37].

However, in some BERT-based models like LaBSE, the second task is replaced with Translation Language Modeling (TLM). This task is an extension of MLM, in which the model receives two concatenated sentences instead of one, where the second sentence is a translation of the first in another language. The rest of the task is the same as in MLM: the whole input is masked, and the model is tasked with predicting the masked tokens [16].

The task layer is used primarily only during pre-training and is omitted when the model is used for a different task, as many use cases do not need tokens in the results but instead use embeddings from the encoding layer as a form of text encoding, which is then used in task-specific layers for fine-tuning.

4.4.2 Pooling layer

After the BERT model returns embeddings for all input tokens, these embeddings need to be aggregated into a single vector that represents the embedding of the entire input. This aggregation is typically performed using a pooling layer. In the case of LaBSE, this pooling is done simply by taking the embedding of the first token, which is the embedding of the special classification token added to the beginning of the BERT input.

4.4.3 Dense layer

The next layer is a standard feed-forward dense layer using a hyperbolic tangent activation function. The number of input and output neurons is the same, and the layer includes an additional bias neuron.

4.4.4 Normalization layer

The final layer is normalization, whose task is to normalize the resulting vector by dividing it by its L_2 norm, ensuring the final vector has an L_2 norm equal to one.

4.5 Word2vec model

Word2vec is a neural network-based method for generating word embeddings, which are dense vector representations of words that capture their semantic meaning and relationships. In other words, properties like the distance between two embeddings contain underlying information about those words, such as their similarity. There are two main approaches to implementing Word2vec:

- Continuous bag-of-words (CBOW)
- Skip-gram

4.5.1 CBOW approach

A model using the CBOW approach receives a sequence of words called the context, with one word missing, and tries to predict the missing target word as output. The model initially assigns one-hot encoding to each word in its dictionary. During training, each word from the context is first converted into its one-hot encoded embedding, which is then multiplied by a weight matrix to obtain its lower-dimensional dense embedding. The dense embeddings of all words in the context are then averaged, and the resulting embedding goes into a hidden layer, which transforms the vector back into the dimension of the vocabulary. Finally, a softmax function is applied to get the probability of each word in the vocabulary being predicted as the missing word. We can see this architecture in Fig. 4.7. Training is usually done using a fixed context window moving along the training text.

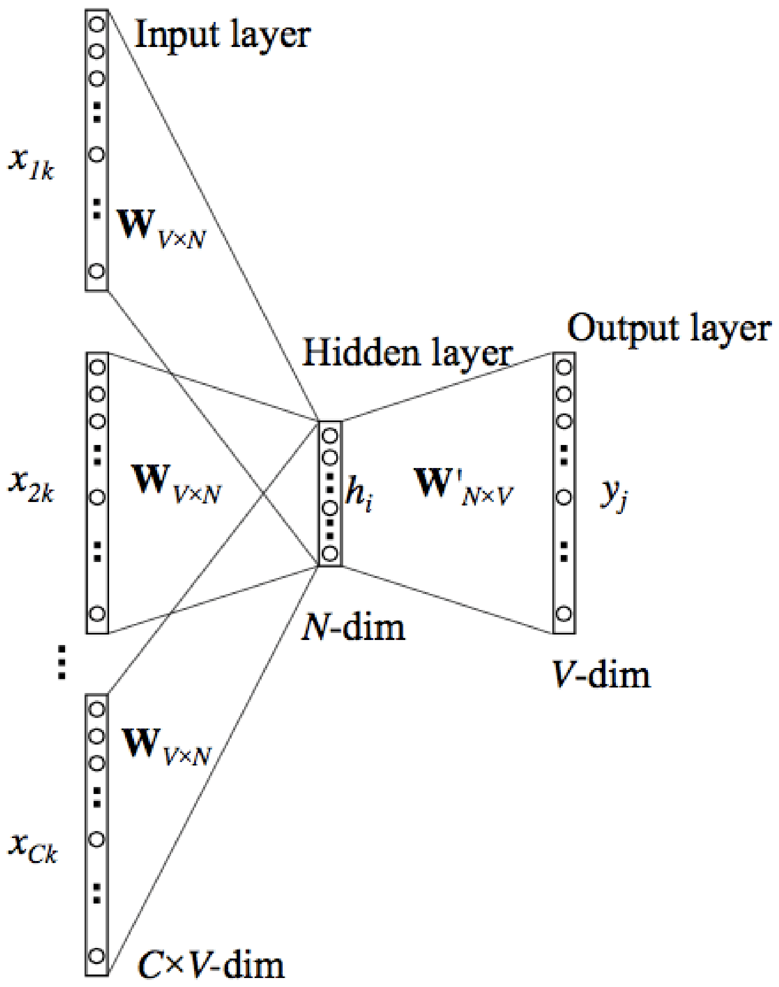


Figure 4.7: Architecture of NN to train CBOW implementation of Word2vec model [24].

4.5.2 Skip-gram approach

The Skip-gram approach works in the opposite way, the model receives a target word and tries to predict the surrounding context words. Similar to CBOW, the input word is first one-hot encoded and then multiplied by a weight matrix to transform it into a dense embedding. This embedding is passed to the next layer, which transforms it back into the vocabulary dimension. A softmax function is then applied to generate a probability distribution over potential context words.

The Skip-gram's loss function is the sum of the negative log-likelihoods of all context words. This architecture is illustrated in Fig. 4.8.

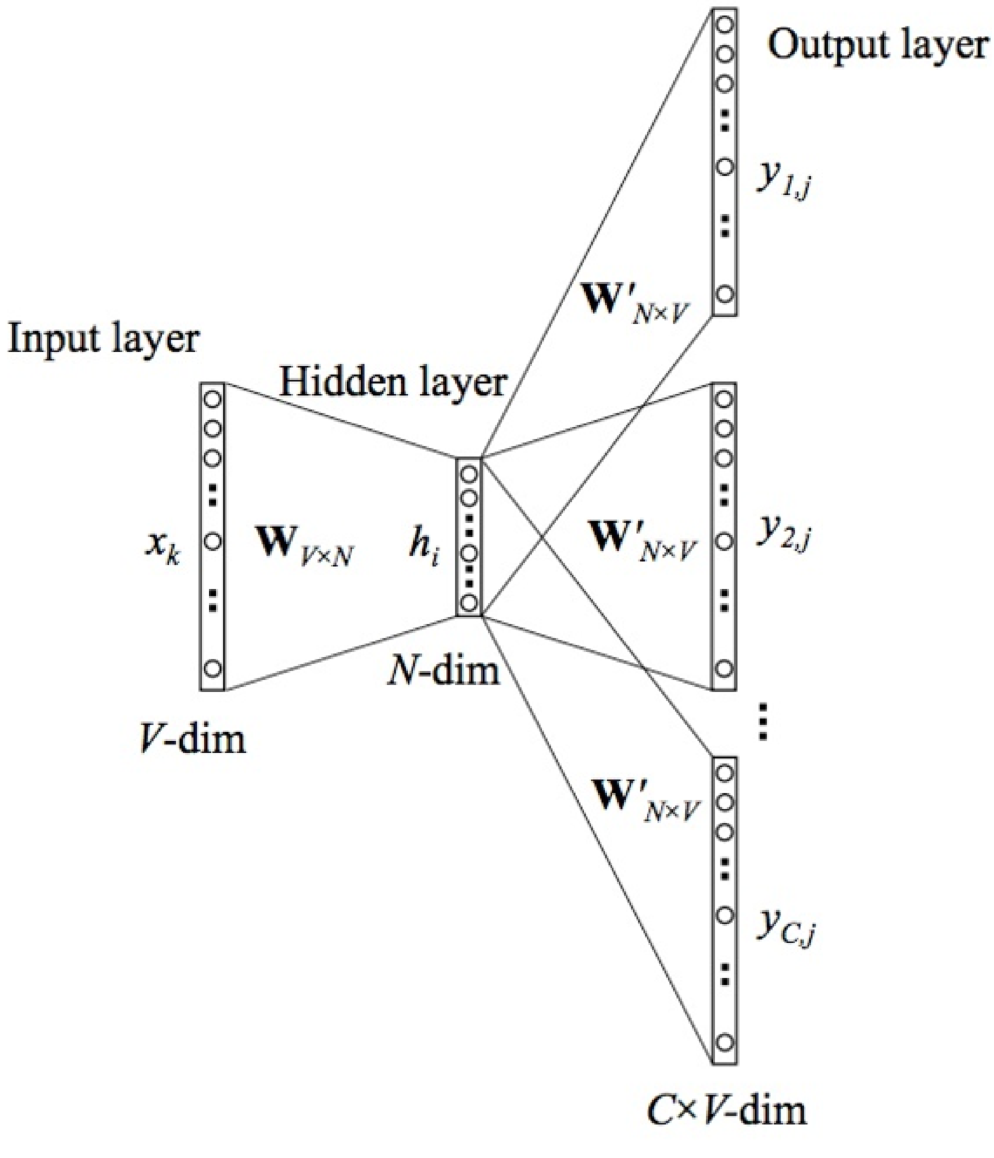


Figure 4.8: Architecture of NN to train Skip-gram implementation of Word2vec model [27]

Chapter 5

Proposed Methods

Task of predicting future cost of a patient can be split input multiple sub-tasks which follow each other. The sub-tasks are these:

1. Embed patient history into numerical vectors
2. Compute expected number of records patient would have in next year
3. Predict future records for patient
4. Predict cost of each future record
5. Complete total cost of patient for next year

5.1 Embedding of Patient

First of sub-tasks for prediction of patient future costs is to embed each patient record into numerical vector that would be understandable for neural network. Our main goal was to create embedding that retain similarity information, meaning that records for similar diagnosis, drugs and medical procedures would receive similar embedding where we define similarity by the Euclidean distance between embedding vectors. Retaining similarity is important for in order to make prediction task of predicting future records easier for model since it would be sufficient to predict just closely related record and not necessary exact one.

For each record of a patient we embed four information. First and easy to implement is a timestamp which is computed using numerical and date values, then there are three more tricky information and those are diagnosis, medical procedure and prescribed drug.

5.1.1 Timestamp

Attribute what we call timestamp of patients record can more precisely described as approximation of patients age at the moment of either medical procedure or drug prescription. This timestamp is computed using two of available information and those are age of patient in years and date of record. The specific procedures we use compute this timestamp can be found in section 7.1.1, in general we find first record, compute timestamp based on patient age as approximate age in days at that point and then compute subsequent timestamp as timestamp of first record plus difference of record dates. This way each timestamp contains approximation of age of patient while also containing information about order of records and comparative timeframe between each two records.

5.1.2 Diagnosis embedding

Base diagnose information we embed was ICD-10-CM code of disease.ICD-10-CM stands for "International Classification of Diseases, Tenth Revision, Clinical Modification" and is used to code and classify medical diagnoses [13] most precisely version of this code that is used in Slovakia and is better known by the acronym MKCH-10-SK (Medzinárodná klasifikácia chorôb) [6].

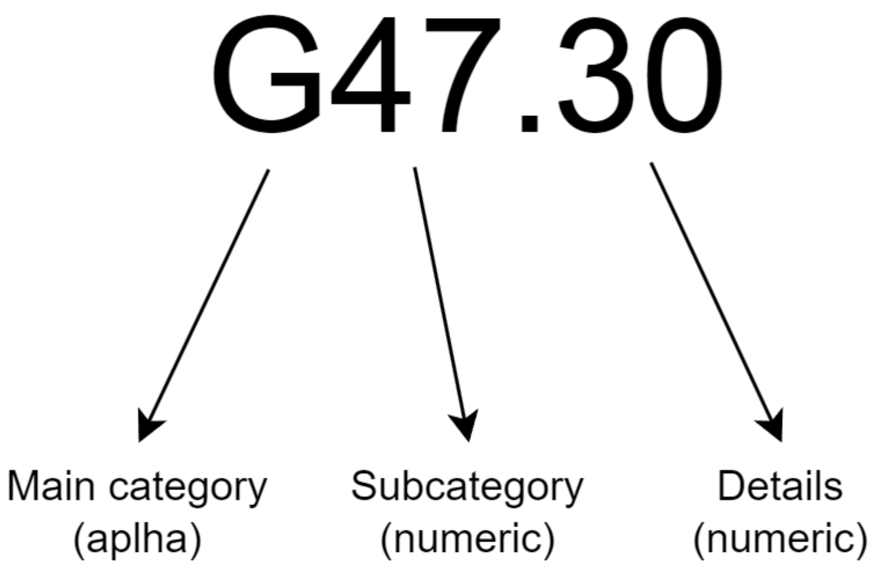


Figure 5.1: Structure of MKCH-10 code.

This code consist of three parts as shown on Fig. 5.1. First part is one letter that encodes main categories of diseases also known as chapter, for example codes starting with G are diseases of the nervous system. After that there is two numeric characters that further specify subcategory of disease such as codes from G40 to G47 which are episodic and paroxysmal disorders and specifically G47 are sleep disorders. We can see that episodic and paroxysmal disorders are only up to G47, meaning that theoretically there can exist subgroups G48 and G49 which have 4 in second position but does not belong to same G4 subcategory as G40 or G47 . Thankfully this is not the case and in cases like this when higher lower subcategory (like G4) does not have 10 lower level subcategories (like G47) these subcategories does not exist at all and in case it have more than 10 lower level subcategories it gets multiple consecutive high level subcategories, for example disorders of other endocrine glands spans from E20 to E35. Then code contains dot after which there are characters that further describe details of disease such as etiology, anatomic site and severity. Official documentation of ICD-10-CM codes stands that codes can be up to 7 characters long meaning that after first three characters specifying category there can be up to 4 alphanumeric to further specify the disease CIT̲E, however Slovak version MKCH-10 codes contains at most two numeric characters to specify disease cit̲e and these details are organized in a way where the first position conveys higher-level information than the second, for example G47.3 is sleep apnea and G47.30 is primary central sleep apnea.

To embed this code we firstly split it into it's three parts and embed each part separately. After that we concatenated embedding of each part to get final embedding of disease. To embed main category we generated vector containing random numbers from uniform distribution for each letter of English alphabet (all main categories). We used random vectors in order to have relatively similar distance between any two main categories since there are no particular relationships between these categories, we were thinking also about using one-hot encoding which would make distance between each two categories perfectly same but we didn't do it since it would have restricted length of vector.

In embedding second part which is subcategory we used different approach, where to each number between 00 and 99 (all possible values of this part) we assigned linearly number in chosen interval. This assigned number was then repeated multiple times to create vector. This approach has advantages and disadvantages. Advantage is that we can be sure that closely related disease subgroups like G46 and G47 would get close embedding since their subgroup codes are close on number line. However there are also two disadvantages, first is that G49 and G50 would be similarly close as G46 and G47, but thankfully in a most cases either X9 code doesn't exist at all, creating a gap, or if X9 code exist it belong to category that go past X on as higher level subgroup. Another disadvantage is that distance between two higher level subgroups can vary quite dramatically even though in reality there might not be reason for that difference in distance. For example, using this approach higher level subgroup G40-G47 is much closer to subgroup G50-G59 than to G80-G83.

Finally to embed details we decided to use same approach as for subgroups since these codes work similarly, only difference was that not all codes had second level details, in such cases we add 5 as a proxy in order to minimize average distance from all potential codes with same first level details code that contain second level detail information while also maximizing average distance to different first level detail codes.

Once all parts are embedded we can create final embedding as their concatenation, to encode importance of each part in final embedding we gave them different lengths, this works thanks to the fact that each value in each of vector have same mean and same variance. Importance of each level was encoded using different lengths of vector for each part where embedding of main category was longest and details got shortest embedding.

5.1.3 Drug embedding

Similarly to diagnosis, to embed drug information we embed international code associated to these drug. In case of drugs it was Anatomical Therapeutic Chemical classification system also known under abbreviation ATC. In a same way as MKCH-10 code this code can be split into multiple parts where each next part contains finer information. It contains of 5 parts or levels. First level encodes main anatomical or pharmacological groups. There are fourteen such groups, encoded by single letter, which are shown in the Fig. 5.2. Then second level encodes pharmacological or therapeutic subgroup using two digit number, after that there two levels that further specify pharmacological, therapeutic or even chemical subgroup, these two levels are both encoded using single letter each. Final fifth encoded with two digit number contains information about specific chemical substance inside drug.

Embedding was done in very similar way as in diagnosis embedding, meaning each level was embedded separately and final embedding was done as concatenations of them. In this case each level was embed using random vector from uniform distribution. We used random vectors for each level since none of levels contains any internal sub-groupings similar to subgroups of diagnosis (see 5.1.2 subgroup codes). To encode importance of levels we again used lengths of random vectors. With this embedding we should get codes whose similarity is more dependent on whether lower more important levels match than higher ones.

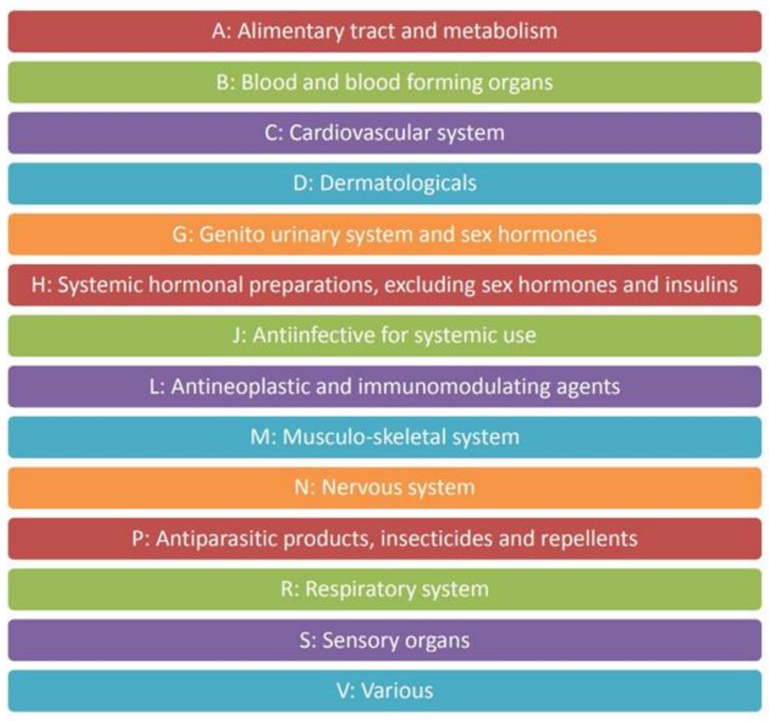

A: Alimentary tract and metabolism
B: Blood and blood forming organs
C: Cardiovascular system
D: Dermatologicals
G: Genito urinary system and sex hormones
H: Systemic hormonal preparations, excluding sex hormones and insulins
J: Antiinfective for systemic use
L: Antineoplastic and immunomodulating agents
M: Musculo-skeletal system
N: Nervous system
P: Antiparasitic products, insecticides and repellents
R: Respiratory system
S: Sensory organs
V: Various

Figure 5.2: Fourteen main anatomical or pharmacological groups and their corresponding first level ATC code [1].

5.1.4 Medical procedure embedding

Final part to embed was medical procedures. In this case there is no structured code that can be used and is implemented in Slovak healthcare systems. So what we have done is to embed description of the procedures. For that we tried two different approaches, first was large language model (LLM) trained on multiple languages including Slovak and second was Word2vec model, explained in section 4.5, specific for Slovak language. Dimensionality of resulting embedding was then reduced using PCA, to get rid of dimensions with small variance meaning they encode small amount of information.

For LLM we specifically choose LaBSE model which we explained in section 4.4, since this model is trained to give similar embedding to similar sentence regardless of language in which sentences are, we hoped that thanks to this approach we would get better embedding of procedure description that contains professional medical terms which are a lot of times international and might not be part of corpus for model trained on single language.

Finally we create record embedding by concatenating all four parts. Since we have two separate datasets, one for prescribed drugs and one for medical procedures, we always have only three out of four information available for each record, since timestamp and diagnosis are always available, we substitute missing part with vector of zeros with appropriate length which is most neutral embedding since we centered both medical procedure and drug prescription embedding around zero.

5.2 Prediction of future number of records

Our task is to get information about how much will cost patient in the future, more specifically, how much it will cost in the next. Since our approach predicts future records, assign to them cost and sum the cost into resulting cost, we need to somehow be able to assess how many records need to be generated in order to simulate approximately next year. For this task we tried two different approaches.

First approach was to predict approximate number of records using linear regression which gets counts of records from previous years and predict next one. Other approach used stopping criterion, which uses timestamp information which is available in each input and generated records, and stop generating once difference between last timestamp from patient data and last generated timestamp surpass one year threshold.

Each of these method have it's own disadvantages. In case of linear regression, number of records per year can vary significantly, so even though we expect increase [10] regression might not be able to catch this trend, especially if patient data consist of only few years in the past. Potential issue with second approach is that it's dependent on how well model learned that timestamp should always increase.

5.3 Future record prediction

This task is most important while also hardest to train. Our goal is to create model which would be able to predict potential next record based on patient previous ones. This task in general is almost impossible with amount of information we have, since there are many other factors that determine whether, when and what new disease will the patient become infected with in the future, whether his current state will improve or worsen, and most importantly what specific steps will doctor take.

Fortunately our ultimate goal is not to predict specific patient future but only estimate how much would he most likely cost. So we expect that even though we wouldn't predict what exactly happens we would be able to estimate total cost of those records.

We try multiple different model for this prediction. First three models we tried were Recurrent Neural Networks (RNN). More specifically we tried multi-layer Elman RNN, multi-layered Gated Recurrent Unit (GRU) RNN and multi-layer Long Short-Term Memory (LSTM) RNN, architecture of these models is explained in Sec. 4.2. Last model we tried was Transformer explained in Sec. 4.3, more specifically we tried Decoder-only Transformer model.

5.4 Record Cost Prediction

Next step in total cost prediction is to predict how much each record would cost. For this we choose standard multilayer perceptron (MLP) or in another words multi-layered fully-connected feed-forward neural network, we explained architecture of this model in Sec. 4.1.

We also tried to train Gradient Boosting model and Ridge model to have a comparison. We choose these models for comparison to have representative from both decision tree and linear regression models, and we choose specifically these representative from each model group since they utilize more complex techniques than base models in their group and they were on par if not better than Artificial Neural Network in similar study by Mohammad Amin Morid at al. [29] which we cover in Chap. 2.

5.5 Prediction of future cost of patient

This last task is our main goal. We wanna be able to predict what would be cost of patient in next year. In order to do this we utilize results from all our previous tasks where we firstly embed patients records after that we predict records that patient could receive in next year, for each predicted record we then predict it record cost category, finally all costs are summed and transformed into patient cost category.

Chapter 6

Software Design

This chapter is dedicated to introduce software used to create, train, validate and used machine learning model described in this thesis. Whole code was written using Python language, more specifically in `Python 3.11.4`. We chose this language for its ease of use and wide selection of libraries for data processing and machine learning. All scripts are available at the GitHub repository `https://github.com/MarianK-py/diploma_thesis_code`.

Whole code can be split into three parts:

1. Embedding - code to add create embedding mapping files
2. Model training and validation - code to setup, train, validate and save prediction models, has to be run once
3. Predictor - code to load trained models and predict future cost of inputted patients

Code for model training and validation, and code for prediction use same technologies, that's we put them into single section. Now we will introduce libraries and pre-trained models used in our code. Firstly ones used in general and then specific ones used in each part mentioned above.

6.1 General

Some of the libraries were used in all parts of code to maintain some coherence in technologies used.

To this category belong `Pandas 2.2.1` library a fast, powerful, flexible and easy to use open source data analysis and manipulation tool [32] which was used to load, manipulate and save all datasets. Another one is `Numpy 1.26.4` library an open source project that enables numerical computing with Python [21] which we use for computations such as random number generation, computations of mean and standard deviation for data normalization and many more.

6.2 Embedding

For embedding we used couple additional libraries since we utilized couple of algorithms and pre-trained models. More specifically libraries and specific algorithms and models we used are these:

- `Scikit-learn 1.2.2` - simple and efficient tools for predictive data analysis [34], we specifically use function to compute PCA in order to decrease dimensionality of medical procedure embedding and function K-means clustering in order to check embedding has desired property.
- `NTLK 3.8.1` - this abbreviation stands for Natural Language Toolkit, it's a library for building Python programs to work with human language data [11], in our case we used tokenizer function to split description of medial procedures into tokens, in our case words.
- `Simplemma 1.1.2` - which provides a simple and multilingual approach to look for base forms or lemmata [9], we used to lemmatize our tokenized text since lemmatizer provided by this library contains also Slovak and Czech languages.
- `SentenceTransformer 2.2.2` - go-to Python module for accessing, using, and training state-of-the-art text and image embedding models [36], which allowed to easily load LaBSE model from Hugging face.
- `Gensim 4.3.3` - library for topic modelling, document indexing and similarity retrieval with large corpora [35], which we used to load Word2vec model trained specifically for Slovak language

6.3 Model training, validation and prediction

For training, validation a using model for predictions we used primarily `PyTorch 2.6.0` which is an optimized tensor library for deep learning using GPUs and CPUs [33]. This library provided us with all required components like linear, non-linear or specialized layers to assemble both simpler neural networks like MLP, we used for prediction of cost category of record, and more complex neural networks like LSTM or Transformer, we used to predict future records. Other then building blocks for networks themselves but also other components required for model training like optimizers and loss function, with possibility to modify them for our specific requirements as we did for loss function used for future record prediction.

Another library used here was `Scikit-learn 1.2.2` from which we used linear regression model, which we consider as one of possible way to know how many future record we should generate for patient to get his expected amount for next year.

Chapter 7

Implementation

7.1 Embedding of Patient

First of sub-tasks for prediction of patient future costs is to embed each patient record into numerical vector that would be understandable for neural network. For each record of a patient we embed four information. First and easy to implement is a timestamp which is computed using numerical and date values, then there three more tricky information and those are diagnosis, medical procedure and prescribed drug.

7.1.1 Timestamp

To compute timestamp we first took all records for single patient and found one with earliest date. This record is then used as pivot for computation of all timestamp for patient. To compute timestamp for this record we took age of patient in years add half year and multiplied it by 365 to get approximate age of patient in days which served as a timestamp, we add half a year in order to improve approximation since we have only age in years meaning we don't know whether they had birthday one or eleven months ago, however we expect it to be on average six months so half of a year. For all subsequent records we compute difference between date of record and date of first record in days and add this difference to timestamp from first record to create one for this record.

7.1.2 Diagnosis embedding

As discussed in Sec. 5.1.2 embedding of diagnose is based on MKCH-10-SK (ICD-10-CM) code of disease. Where we split this code into three parts and embed each other independently and finally concatenate result.

To embed main category we generated vector containing random numbers using uniform distribution for each letter of English alphabet we choose to sample these random numbers from interval [-0.5, 0.5]. We choose this interval mostly arbitrarily since we were planing to pass recording embedding into normalization function once it complete.

For subcategory and details we linearly assigned value to each possible two digit code. We chose interval from which we take this value to be [-0.5, 0.5], meaning subcategory 00 would get -0.5 category 50 would get 0 and category 99 would get 0.5, this interval was chosen in order to for each dimension of this embedding to have same mean and standard deviation as each dimension of main category. Thanks to that they also have on average same distance per dimension. This means that each position of each part of embedding should contribute to total distance with same weight.

Most important part was to assign lengths to vector of each part in a way that would encode their importance. Main category, the most important part, got vector of length 28, subcategory part got length 7 and finally details got length 3.

Showcase of resulting embedding can be seen on Fig. 7.1 where each part is highlighted by different color and all values are rounded to two decimal.

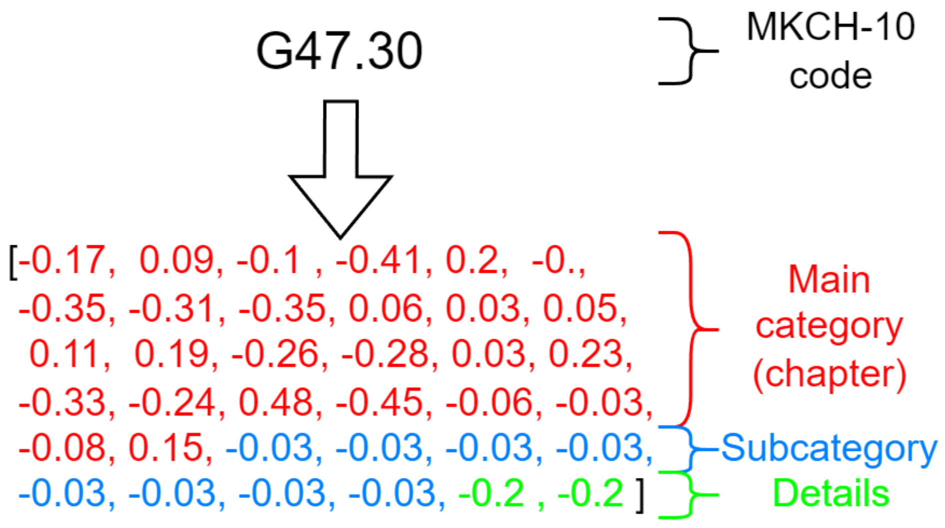


Figure 7.1: Showcase of resulting embedding of specific diagnosis (rounded to two decimal places).

7.1.3 Drug embedding

Embedding was done in very similar way as in diagnosis embedding, meaning each level was embedded separately and final embedding was done as concatenations of them. In this case each level was embed using random vector from uniform distribution with interval [-0.5, 0.5]. We used random vectors for each level since none of levels contains any internal sub-groupings similar to subgroups of diagnosis (see Sec. 5.1.2 subgroup codes). To encode importance of levels we again used lengths of random vectors, where with higher level vectors shortens. Lengths of vectors for each level can be seen in Tab. 7.1. So total length of embedding is same as embedding for diagnosis.In case code is incomplete, meaning it's missing higher levels, missing part is substituted with zeros which we consider neutral elements.

Level	Length
1	21
2	9
3	5
4	2
5	1

Table 7.1: Lengths of random vectors assigned to each information level of ATC code.

With this embedding we should get codes whose similarity is more dependent on whether lower more important levels match than higher ones.

7.1.4 Medical procedure embedding

Embedding of medical procedure was straightforward since we used already trained model.

As discussed in 5.1.4, for LLM we choose LaBSE model. It's a model developed by Goggle to encode text into high dimensional vectors. This model was trained 109 languages including Slovak. Using this model was straightforward and we just had to input complete description of procedure into to receive 768 dimensional dense encoding of it. After computing all embedding we performed principal component analysis (PCA) to reduce dimensionality of this embedding while maintaining most of the variance or in other words most of the information stored inside it.

We tried also different approach using Word2vec model trained specifically for Slovak language. More specifically we used word2vec-sk model made by company Essential Data [2]. This model was trained on corpus containing around 110 millions of words. More specifically we choose version of model trained using CBOW approach. Since this model is trained to embed words not a text, we firstly split description of procedure into words and lemmatize those words, in other word change into their base form, then using Word2vec model we embed each word into dense 200 dimensional vector separately and finally create description embedding as an average of embeddings of all words in it.

We expected that LaBSE model produce better results compared to standard text embedding models trained solely on Slovak language, since LaBSE model is during training comparing embedding not only to similar sentences in Slovak language but also their translation in other which could mitigate a relatively small amount of Slovak language data compared to other more commonly used languages. Additionally this model could know domain specific words in our case medical terms which are left in foreign language and would most likely not be found in Slovak only corpus.

Finally we create record embedding by concatenating all four parts. Since we have two separate datasets, one for prescribed drugs and one for medical procedures, we always have only three out of four information available for each record, since timestamp and diagnosis are always available, we substitute missing part with vector of zeros with appropriate length which is most neutral embedding since we centered both medical procedure and drug prescription embedding around zero.

7.2 Prediction of record cost

As discussed in 5.4 we would use MLP to predict cost of record. However since our goal is to predict future cost category of patient and not exact cost amount we decided to also predict cost category of record instead of it's precise cost.

To assess how to split interval of possible costs into sub-intervals corresponding to each category with visualized costs of records from our training dataset in histogram. Resulting histogram can be see on Fig. 7.2 where y axis is linear while x axis is logarithmic as well as size of bins increase logarithmically, there we can see that most of records have cost between 0.1€ and 200€, so that's where we want most of cost categories to be. We decided to merge all cost under one euro into single category since even though they are numerous they have very small influence on total cost of patient in year. Then we split interval between 1 and 200 into 6 categories with increasing width, after that we group few outliers between 200 and 500 into one category and finally we give all outliers above 500 last category. This gave us 9 categories which are summarized in Tab. 7.2.

Model itself consist of input layer of size 196 (final size of our embedding) after that there are multiple linear layers with non-linear functions in-between. In then with get to final layer of size 9, so number of our categories, results of this final linear layer goes into softmax function to get transformed from resulting values into probabilities

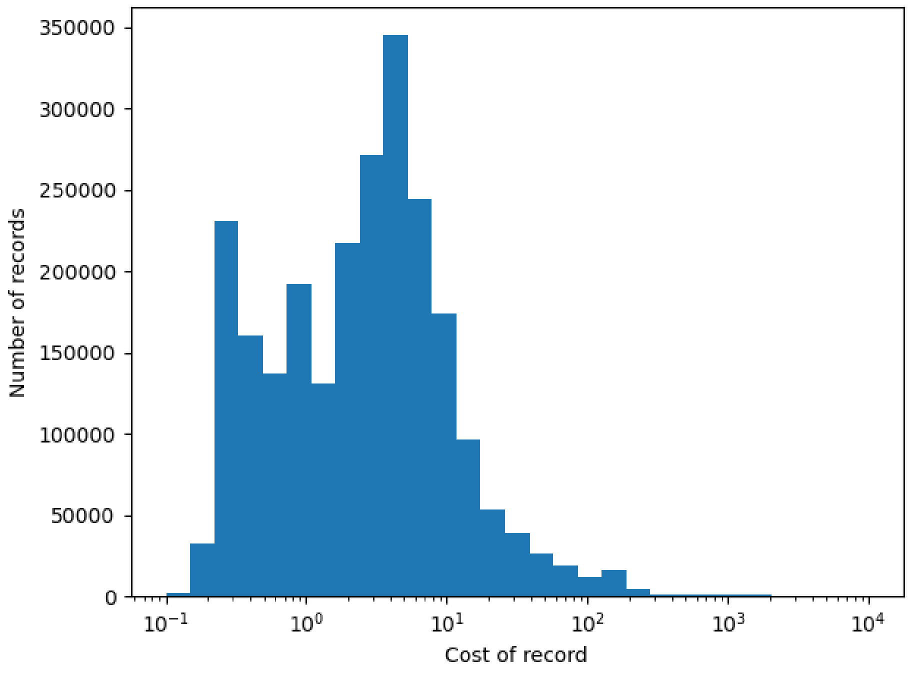


Figure 7.2: Histogram showing distribution of cost of records in training dataset.

Category	Interval
1	[0,1)
2	[1,5)
3	[5,10)
4	[10,20)
5	[20,50)
6	[50,100)
7	[100,200)
8	[200,500)
9	[500,∞)

Table 7.2: Intervals of record cost for each category.

of each category.

We optimize multiple parameters of network itself as well few parameters important for training. From perspective of model itself we tried multiple number of layers so depths of model, as well as their size and non-linear functions in-between them. From training perspective tried three loss functions, more precisely we tried mean square loss, cross entropy loss and negative log likelihood loss, and multiple values for learning rate. For optimizer we choose Adaptive Moment Estimation (Adam) [23], which is an optimizer that deliver a powerful method for adjusting the learning rates of parameters during training and can be consider industry standard at this point.

Setting of parameters was done by training multiple models on smaller subset of dataset locally and then final model was trained on chosen hyperparameters with complete dataset on server. All model were trained using using batch approach to limit amount of data loaded at once while not computing gradient solely on loss from single input, in our case single patient record.

7.3 Prediction of future records

For task of prediction of future records of patient based on their history we in the decided to try two models, those models were LSTM and Transformer, we very briefly tried simpler models like basic RNN and GRU but in both cases we immediately significantly worse results so we decided to omit them from our results entirely and focus on more complex models.

In both tested models we optimized depth of model, meaning how many hidden layers model consist of, and dropout rate, meaning what percentage of neurons we randomly shut down, in other words set to zero, at each training step in order to let other neurons partially learn patterns from shut down neurons which reduce overfitting and hence it help model get more generalized. In case of LSTM model we additionally optimized size of hidden layer and as for Decoder-only transformer we optimized number of transformer heads.

Since our embedding of patient record consist of multiple parts that each have different length we decided to create our own modification of loss function that would take length of each part into consideration in order to give them same weight or in other words importance. We decided to base this loss function on mean square error (MSE) loss function which computes loss using this formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y_i})^2. \tag{7.1}$$

Where n is number of dimensions, Y_i is target value at i-th dimension and $\hat{Y_i}$ is value predicted by model at that dimension. Using this approach each single dimension have same importance, meaning that if our vector would have 30 dimensions and would consist of two parts, one with 20 and other with 10 dimensions, first one would get twice as much importance purely because of higher number of dimensions. We decided to solve this issue by computing MSE loss of each part, and then computing mean of these losses to get final loss. We call this modified loss function subpart weighted MSE as it adjust weights of each dimension in a way to give same way to each subpart of vector, formula for this loss can be written like this:

$$SubpartWeightedMSE = \frac{1}{p}\sum_{j=1}^{p}(\frac{1}{l_j}\sum_{i=1}^{l_j}(Y_{s_j+i} - \hat{Y_{s_j+i}})^2). \quad (7.2)$$

Where we can see couple of additional parameters, p is number of subparts, l_j is number of dimensions of j-th subpart and s_j is index of first dimension of j-th subpart. For optimizer we used same one as in cost of record prediction model, which is Adam.

When we begin trying to train these model we came across memory limitations, so we decided to limit context window, meaning number of past record seen by model. Whole process of data preparation for training consist if this sequence of steps:

1. Get batch of patients.
2. For each patient in patch:
   (a) Get all records for patient.
   (b) Order records of patient.
   (c) Using moving window of our maximal context size, that moves by one record at the time, this generate list of all windows for patient.
   (d) We generate input list by removing last window and target list by removing first window.
3. Collect inputs and targets for all patients in batch.

In step 2. (b) we order primarily order by date in ascending order and if we have multiple records with same date then secondarily we order them based on internal ID value of diagnosis, medical procedure and drug in this order, we do this to make sure ordering is coherent throughout all training inputs.

By generating input and target list as explained in step 2. (c) we use i-th window as an input we expect $(i+1)$-st on output. We did this to increase amount of information model can learn by asking model to give prediction from incomplete context, meaning that if we have context of size n we do not only want to predict $(n+1)$-st record but also for all k such that $0 < k < n$ we want to predict $(k+1)$-st record based on first k records from context. Both our tested model support this behavior, in case of LSTM model it's integrated directly in model backbone provided by `Pytorch` library since we are not using bidirectional version of model [5], and in order to achieve it in Decoder-only transformer model we used causal masking inside self-attention layers which forbids queries to see keys from future based on it's perspective.

Similarly to cost prediction models, these models were also firstly trained on smaller subset of data to assess best values for hyperparameters being optimized and best one was then retrained on complete dataset on server. Another similarity is that we used batch training for these models as well, however in this case since transforming patient record into lists of windows cause them to increase in memory size we use two types of batches, first is patient batch, which is group of patients that got their records transformed together and then training batch, which is usual type of batch so number of windows that goes into model at once.

Chapter 8

Research

This chapter is dedicated to determining the most suitable parameters for embeddings and prediction model in a way that resulting embeddings would satisfy requirement to give similar embedding to similar inputs and resulting model have lowest loss and highest accuracy.

8.1 Embedding of patient

First of sub-tasks for prediction of patient future costs is to embed each patient record into numerical vector that would be understandable for neural network. For each record of a patient we embed four information. First and easy to implement is a timestamp which is computed using numerical and date values, then there three more tricky information and those are diagnosis, medical procedure and prescribed drug.

8.1.1 Diagnosis embedding

We need to confirm that closely related diagnosis would get embedding with smaller distance meaning higher similarity compared to less related diagnosis. In order to confirm that our embedding has desired properties we computed similarity of embedding of multiple codes. As similarity function we choose simple multiplicative inverse of Euclidean distance. In Tab. 8.1 we can see results. Highest similarity was between codes G47.30 and G40.09 which is expected since they belong to same main category and very close subcategory, second highest was between H40.09 and H18.80 which are only other combination that belong to same main category, this confirms that main category has biggest impact since this similarity is significantly higher than that between G40.09 and H40.09 which differ only main category.

Another confirmation that embeddings works as intended by clustering them. More specifically we would expect that if we group embeddings into as many clusters as are main categories, each cluster should contain mostly if not only diagnosis from one main

Code A	Code B	Similarity
G47.30	G40.09	2.77
G47.30	H40.09	0.53
G47.30	H18.80	0.46
G40.09	H40.09	0.54
G40.09	H18.80	0.45
H40.09	H18.80	0.84

Table 8.1: Similarities of embedding of multiple chosen MKCH-10 codes

category. On order to test this we used K-means clustering with 26 clusters which is number of different main categories of diagnosis and got these results:

Cluster ID	Cluster size	Frequency of first level values
0	654	C: 654,
1	2092	M: 2092,
2	766	Y: 766,
3	543	S: 543,
4	786	Z: 786,
5	1100	T: 1100,
6	1067	X: 1067,
7	620	H: 496, U: 124,
8	752	S: 752,
9	1770	M: 1770,
10	739	Q: 739,
11	584	D: 584,
12	507	F: 507,
13	958	W: 958,
14	556	E: 556,
15	491	A: 491,
16	553	O: 553,
17	627	K: 627,
18	872	V: 872,
19	521	G: 521,
20	463	L: 463,
21	420	R: 420,
22	465	B: 465,
23	717	J: 319, P: 398,
24	568	I: 568,
25	535	N: 535,

Table 8.2: Size of clusters and frequencies of first level diagnosis codes in them.

In Tab. 8.2 we can see that almost every cluster consist of diagnosis from with only single main category. Only immediately visible issues are clusters 7 and 23 where two main categories got assigned same cluster which is most likely caused by their small size compared to other and by the fact that largest categories M and S got split into two clusters.

8.1.2 Drug embedding

To confirm this we do similar check as we did with diagnosis code and compute similarity of four chosen ATC codes and see if our theory holds. To compute similarity we again use multiplicative inverse of Euclidean distance. We choose C01EB15, C01CA04, C10AA07 and J01CA04, we would expect first two to be most similar since they match on first levels, than first two compared to third should have slightly lower similarity since they match on only first level and finally we expect that all three would be least similar to fourth one since first level is different, even though that second and forth match on all other levels. We can see results in Tab. 8.3. We can see that results met our expectation with highest similarity between first two and lowest between any of first three and fourth, even in case of second and forth that match on all other levels except first.

Code A	Code B	Similarity
C01EB15	C01CA04	1.24
C01EB15	C10AA07	0.54
C01EB15	J01CA04	0.45
C01CA04	C10AA07	0.64
C01CA04	J01CA04	0.48
C10AA07	J01CA04	0.38

Table 8.3: Similarities of embedding of multiple chosen ATC codes

Also similarly to diagnosis we can test whether our embeddings would cluster mainly first level of code or not. Since there are 14 main anatomical or pharmacological groups as shown on Fig. 5.2 we used K-means algorithm with 14 clusters and got these results:

Based on results we can see on Tab. 8.4 we concluded that embeddings works generally as intended. We can see that clusters mostly consist of embeddings with same first level values. There are some exceptions which are most likely caused by uneven number of drug in each category. Because of this categories with smallest number of drug like P, S, D, H or G got assigned to cluster along with bigger category and categories with biggest number of drugs like N, C or V got split into multiple categories.

Based on results of these tests we can conclude that embeddings works generally as intended.

Cluster ID	Cluster size	Frequency of first level values
0	8645	C: 8645,
1	19132	N: 19132,
2	9322	L: 8657, S: 665,
3	11734	A: 11734,
4	7159	B: 7159,
5	9526	V: 9526,
6	4681	R: 4681,
7	14732	C: 14732,
8	7237	V: 7237,
9	4031	M: 3987, P: 44,
10	3599	G: 3599,
11	2367	N: 2367,
12	9032	D: 1266, G: 897, H: 1136, J: 5733,
13	6093	N: 6075, P: 18,

Table 8.4: Size of clusters and frequencies of first level drug codes in them.

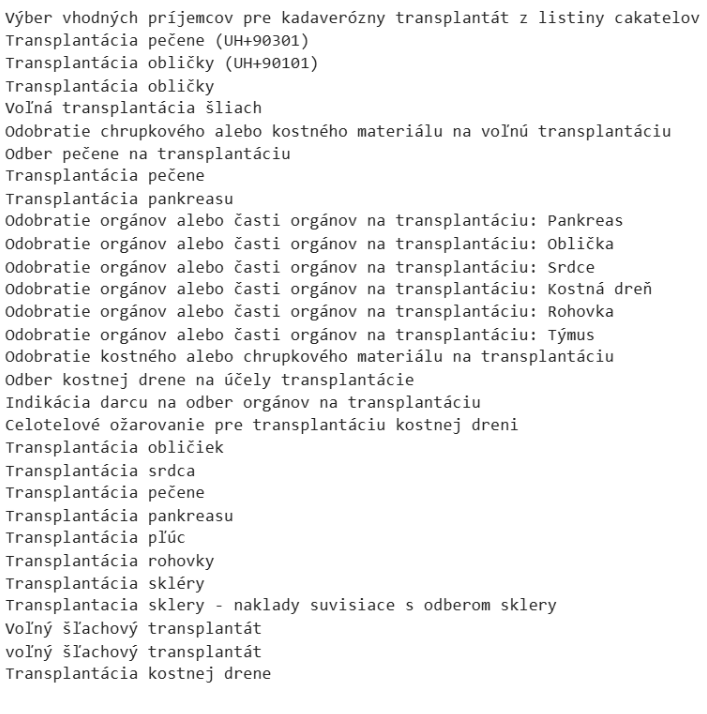
8.1.3 Medical procedure embedding

As said in previous chapters we tried two different approaches both using pretrained models. After we embed medical procedure descriptions both ways we quickly found out that a lot of procedures did not received any embedding from Word2vec model, more specifically 731 out of 7329 procedures, so almost 10% were not embedded, and even among procedures that receive any kind of embedding there were a lot words that were not embedded and so they didn't contributed to resulting embedding. This was caused mostly by limitations of Word2vec model that is able to embed only words which are in it's dictionary, so words available in it's training corpus, and it seems like a lot of professional medical terms such as 'polycystické' or 'cytokín' were not in corpus. Similar issue potentially could have happened also while using LaBSE model, however we found no case where LaBSE model was not able whole procedure. Also thanks to most professional medical terms being similar across multiple languages and fact that LaBSE model was trained on much bigger corpus consisting of 109 distinct languages it's highly probable that much more terms were successfully embed creating much better embedding of whole description.

Since results of embedding using Word2vec model has a lot of issue we decided to proceed using only LaBSE model which was at least able to embed all descriptions. This model gave us embedding with 768 dimensions, in order to densify this information while maintaining most of it we decided to use PCA. We specifically wanted to first few dimensions that would explain over 90% of variance, since in the case of PCA, the fraction of explained variance is often used as a good proxy for the fraction of pre-served information. After running this algorithm we ended up with 119 dimensional embedding that explained 90.4% of variance of original embedding.

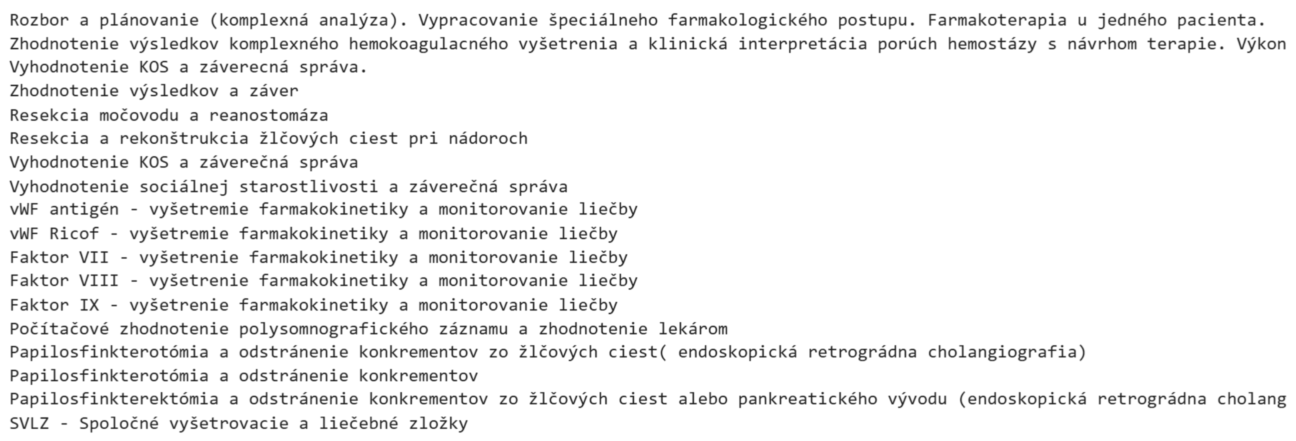
To confirm that resulting embedding successfully encode description of procedure and gave similar procedures similar embedding we did same thing as in case of drug and disease embedding validation and try to group embeddings using K-means algorithm. However in this case we did not have any information similar to highest level of codes for drugs and diagnosis on which we could base our validation, so we decided to just manually look through clusters to assess whether procedures inside them have meaningful connections.

Since we have 7329 procedures we choose our K, so number of clusters, to be 365 to have on average close to 20 procedures in single group. This decision was mostly arbitrary. After checking multiple clusters we found cases such as group cluster number 215 which procedures are listed in Fig. 8.1 where we can see that all procedures has something to do with transplantation. However in some cases clusters contained mixture of seemingly random procedures, such as procedures in cluster 45 in Fig. 8.2 where we can see mixture evaluation of final reports mixed with investigation of pharmacokinetics and papillosphincterotomy, fortunately it seems content of clusters like these is not purely random but consist of multiple subgroups which might indicate embedding works as intended and we just clustered procedures into too few groups.



```
Výber vhodných príjemcov pre kadaverózny transplantát z listiny cakatelov
Transplantácia pečene (UH+90301)
Transplantácia obličky (UH+90101)
Transplantácia obličky
Voľná transplantácia šliach
Odobratie chrupkového alebo kostného materiálu na voľnú transplantáciu
Odber pečene na transplantáciu
Transplantácia pečene
Transplantácia pankreasu
Odobratie orgánov alebo časti orgánov na transplantáciu: Pankreas
Odobratie orgánov alebo časti orgánov na transplantáciu: Oblička
Odobratie orgánov alebo časti orgánov na transplantáciu: Srdce
Odobratie orgánov alebo časti orgánov na transplantáciu: Kostná dreň
Odobratie orgánov alebo časti orgánov na transplantáciu: Rohovka
Odobratie orgánov alebo časti orgánov na transplantáciu: Týmus
Odobratie kostného alebo chrupkového materiálu na transplantáciu
Odber kostnej drene na účely transplantácie
Indikácia darcu na odber orgánov na transplantáciu
Celotelové ožarovanie pre transplantáciu kostnej dreni
Transplantácia obličiek
Transplantácia srdca
Transplantácia pečene
Transplantácia pankreasu
Transplantácia pľúc
Transplantácia rohovky
Transplantácia skléry
Transplantacia sklery - naklady suvisiace s odberom sklery
Voľný šľachový transplantát
voľný šľachový transplantát
Transplantácia kostnej drene
```

Figure 8.1: Medical procedures grouped in cluster number 215.



```
Rozbor a plánovanie (komplexná analýza). Vypracovanie špeciálneho farmakologického postupu. Farmakoterapia u jedného pacienta.
Zhodnotenie výsledkov komplexného hemokoagulacného vyšetrenia a klinická interpretácia porúch hemostázy s návrhom terapie. Výkon
Vyhodnotenie KOS a záverecná správa.
Zhodnotenie výsledkov a záver
Resekcia močovodu a reanostomáza
Resekcia a rekonštrukcia žlčových ciest pri nádoroch
Vyhodnotenie KOS a záverečná správa
Vyhodnotenie sociálnej starostlivosti a záverečná správa
vWF antigén - vyšetremie farmakokinetiky a monitorovanie liečby
vWF Ricof - vyšetremie farmakokinetiky a monitorovanie liečby
Faktor VII - vyšetrenie farmakokinetiky a monitorovanie liečby
Faktor VIII - vyšetrenie farmakokinetiky a monitorovanie liečby
Faktor IX - vyšetrenie farmakokinetiky a monitorovanie liečby
Počítačové zhodnotenie polysomnografického záznamu a zhodnotenie lekárom
Papilosfinkterotómia a odstránenie konkrementov zo žlčových ciest( endoskopická retrográdna cholangiografia)
Papilosfinkterotómia a odstránenie konkrementov
Papilosfinkterektómia a odstránenie konkrementov zo žlčových ciest alebo pankreatického vývodu (endoskopická retrográdna cholang
SVLZ - Spoločné vyšetrovacie a liečebné zložky
```

Figure 8.2: Medical procedures grouped in cluster number 45.

8.2 Prediction of record cost

In case og prediction of record cost category we planned to use MLP model. All models we tested have in common that on input they took 196 dimensional vector, which is dimensionality of out embedding, after that there was multiple hidden linear layers with non-linear functions between them with final layer outputting 9 dimensional vector, which is number of cost categories, followed by softmax function to transform this vector into probabilities of each category. For purpose of loss computation we compared correct category to raw result from model while for accuracy computation we firstly transform this result into one-hot vector based on maximal value in resulting vector before comparing to correct category.

Each model that was trained to assess best parameters was trained on subset of dataset containing 800 patient for training and 200 for validation. In terms of number of records, training dataset consisted on 2 132 436 records while validation contained 533 110 records.

As mentioned in Sec. 7.2 we were mainly focused on assessment of best depth, layer sizes and non-linear activation functions in-between those layers, but we also tested multiple different loss functions. Since number of possible depths and combinations of sizes are activation function we decided to approach it this way. For number of layers meaning depth we tried model with depth we denote as 0, which means that model contains single layer to transform vector of input dimension to vector of categories dimension followed by softmax function, we used this as a base value to see if there is something model can learn better from data, then we tried models width depths denoted as 1, 3, and 6, which means that in-between input vector and layer to result in category vector there are 1, 3 or 7 additional layers with their own activation functions after them. We choose these values to have representation of shallow model, slighly deeper one and relatively deep one. For each non-zero additional depth we tried multiple different configurations to see if we get significantly different results. We tested three configuration types from perspective of layer sizes, in first one we just gradually decrease layer size from input size to output size, in second which we firstly increase size from input to let it learn more patterns in input and then decrease it down to output size and in last we leave size to be of an input unlit very last layer which decrease to output size. For each of these layer size configuration we tested two non-linear activation function configurations, first being only Gaussian Error Linear Units function (GELU) in-between each layer and second being random combination of different activation function picked from this list:

- Sigmoid function (Sigmoid)
- Hyperbolic tangent function (Tanh)
- Rectified linear unit function (ReLU)
- Leaky Rectified Linear Unit function (LeakyReLU)
- Gaussian Error Linear Units function (GELU)
- Sigmoid Linear Unit function (SiLU)

We choose these functions to have mixture of more standard functions like Sigmoid function and more modern varieties like Sigmoid Linear Unit function (SiLU). This in total gave us 28 different models (3 layer size configurations multiplied by 3 activation function configuration multiplied by 3 different non-zero addition depths plus 0 additional depth base model). We did this testing three times with three different loss functions so in total 84 models. Resulting accuracies are compiled in Tab. 8.5.

First loss function we tested was mean square error (MSE) loss, which computes loss based on equation shown in Eq. 8.1 where n denotes number of samples, Y_i it i-th target vector and $\hat{Y_i}$ is i-th predicted vector, this gives result for each dimension which is then reduced one more time also using mean to get single number as a loss. When we look at results for this loss function in Tab. 8.5 we can see that adding layers improves model accuracy but only to certain point since model with only single layer followed by softmax function resulted in lowest accuracy barely just above 60% while almost all other models surpassed 70%, however as we deepen model further we get to point of diminishing return as we can almost no improvement once model has 3 or more layers and accuracy slowly approach 80%. We can also see no signs of overtraining since in all cases difference between training and validation loss and accuracy is very small. From perspective of layer size architecture we can see that we received best usually received best results in models which firstly expand dimensionality and then decrease it to the number of categories.

$$Loss = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{8.1}$$

Second loss function we tested was cross entropy loss, which in theory should be better for training classification problem like ours. We can see interesting behavior that very deep model perform similarly or even worse than shallow one and model with depth in-between them performed the best, this could be potentially caused by issues like vanishing gradient. Other than that from perspective of layer sizes and activation function models with this loss function show similar behavior to models with MSE loss. In general we can see that accuracy of these models is almost always lower compared to models with MSE loss.

Last but not least loss function we tried was negative log likelihood (NLL) loss which should similarly to cross entropy loss be useful to train a classification problem. In this case results are very similar compared to MSE loss in all regards, with only small difference being slightly lower accuracy is less deep models however in deepest model tested we see relatively small to almost no tangible difference.

Based on these results we decided that for training on complete dataset we would use model from row 26 from Tab. 8.5, so 8 layered model (7 additional layers and final layer with softmax activation function) with layer sizes and activation functions as in table and MSE loss which boasts highest accuracy in our testing. Since layer sizes, activation functions and depth were chosen arbitrarily we did some additional testing to tweak these parameters further however we did not find setup which would have tangibly higher accuracy so we decided to continue with current setup.

Finally before moving to main training we compared approach using MLP to two other types of classifiers. First was Gradient Boosting Classification Tree model for which we tried to train trees with maximal depth of 1, 5, 10 and 20. Second tried classifier was Ridge regression model where we tried multiple values of parameter alpha which denotes regularization strength, we tried values $10^{-10}, 10^{-5}, 0.1, 1, 10, 10^3$.

We can see results of Gradient Boosting model in Tab. 8.6. There we can notice initial increase of accuracy however beyond depth of 10 we see no improvement. Best model looks to be model with maximal depth of 10 which has around 71% which is significantly lower than 78% accuracy of our best MLP model.

Depth	Layer sizes	Activation functions	Mean square error		Cross entropy		Negative log likelihood	
			Test accuracy	Validation accuracy	Test accuracy	Validation accuracy	Test accuracy	Validation accuracy
0	[]	[]	63.2%	63.0%	63.5%	63.3%	62.9%	62.6%
1	[98]	[GELU]	74.5%	74.3%	72.8%	72.6%	73.2%	73.0%
		[Tanh]	71.3%	71.0%	70.9%	70.6%	71.1%	70.9%
	[392]	[GELU]	76.6%	76.3%	74.2%	74.0%	75.5%	75.2%
		[Sigmoid]	70.8%	70.6%	69.9%	69.6%	70.2%	70.0%
	[196]	[GELU]	75.9%	75.6%	74.1%	73.8%	74.1%	73.8%
		[LeakyReLU]	76.0%	75.8%	75.5%	75.3%	74.1%	74.0%
3	[98, 48, 24]	[GELU, GELU, GELU]	74.4%	74.2%	72.0%	71.8%	72.2%	72.0%
		[Sigmoid, ReLU, Tanh]	69.7%	69.5%	69.7%	69.4%	70.3%	70.1%
	[392, 196, 98]	[GELU, GELU, GELU]	77.7%	77.5%	75.2%	75.0%	75.5%	75.3%
		[SiLU, ReLU, GELU]	77.4%	77.2%	74.8%	74.5%	75.2%	75.0%
	[196, 196, 196]	[GELU, GELU, GELU]	77.2%	77.0%	74.7%	74.3%	75.0%	74.7%
		[ReLU, Sigmoid, SiLU]	76.7%	76.5%	73.9%	73.7%	75.0%	74.8%
7	[142, 104, 72, 48, 30, 18, 12]	[GELU, GELU, GELU, GELU, GELU, GELU, GELU]	77.1%	76,9%	72.0%	71.7%	76.5%	76.2%
		[SiLU, Sigmoid, GELU, Tanh, ReLU, Sigmoid, LeakyReLU]	75.1%	75.0%	70.3%	70.1%	75.1%	74.9%
	[588, 294, 147, 49, 98, 36, 18]	[GELU, GELU, GELU, GELU, GELU, GELU, GELU]	77.6%	77.4%	71.2%	71.0%	77.4%	77.2%
		[SiLU, GELU, Sigmoid, GELU, SiLU, GELU, LeakyReLU]	78.3%	78.0%	72.0%	71.8%	76.9%	76.7%
	[196, 196, 196, 196, 196, 196, 196]	[GELU, GELU, GELU, GELU, GELU, GELU, GELU]	77.3%	77.1%	72.8%	72.6%	77.4%	77.2%
		[Sigmoid, GELU, ReLU, SiLU, LeakyReLU, GELU, SiLU]	77.1%	76.9%	72.8%	72.6%	77.2%	77.0%

Table 8.5: Accuracies for various MLP models using different loss functions.

Maximum depth	Train accuracy	Validation accuracy
1	51.5%	51.4%
5	69.0%	68.8%
10	71.5%	71.4%
20	71.2%	71.0%

Table 8.6: Accuracies of Gradient Boosting models with different maximal depths of tree.

Results of Ridge model testing are shown in Tab. 8.7 where we can see that differ-

Alpha	Train accuracy	Validation accuracy
10^{-10}	67.1%	66.9%
10^{-5}	67.0%	66.8%
10^{-1}	67.0%	66.8%
10^{0}	67.0%	66.8%
10^{1}	66.9%	66.7%
10^{3}	66.8%	66.7%

Table 8.7: Accuracies of Ridge models with different regularization strength.

ent regularization strengths has no tangible effect on model performance, there seems to be very small decrease of accuracy with increased strength but not significant. All models have accuracy around 67% which is higher then most basic MLP or Gradient Boosting models but is easily outperformed by their deeper versions.

We can see that neither of models used for comparison was able to outperform our MLP model, so we saw no reason to use any of these models instead and stayed with our best MLP model, which was assessed earlier, for main training with 78.3% training accuracy and 78.0% validation accuracy, training loss of this model was 0.0346 and validation loss was 0.0349.

8.3 Prediction of future records

As said in 7.3 we tried two type of models and those are standard RNN models and Decoder-only Transformer. In both cases we tried multiple combination of few hyper-parameters to get most successful model.

We trained each model on same subset of data which consist of 100 patient, which have in total (get number of records) records to train model for 5 epochs and then validate each model on data from 10 patient which are different from one used in training and have (get number of records) records in total. This number is significantly lower to assessment of parameters for model to predict cost of record this training of these models was significantly more complex from perspective of both computational and memory complexity.

Firstly we have to assess which of standard RNN models we wanna test, whether for our task would basic Elman RNN be sufficient or whether we would see tangible improvement by using more complex model like GRU or LSTM. For this we tried all three models with same setup. First setup we tested was 6 layers with width 196 and 20% dropout rate for regularization. With this setup we received results shown in Tab. 8.8 where we can see that LSTM model achieved lowest loss on both training and validation data, with Elman RNN having second lowest training loss while GRU having second lowest validation loss. Based on these results we could conclude that LSTM model bring tangible improvement.

Model	Train loss	Validation loss
Elman RNN	0.5757	0.6689
GRU	0.6278	0.6517
LSTM	0.5156	0.6328

Table 8.8: Comparison of RNN models with 6 layers of width 196 and 20% dropout rate.

To make sure these results are not fluke we decided to test one more setup. This time tried significantly bigger model deepened to 12 layers, so twice the depth, each with width 784, so quadruple width, we still left dropout rate to be 20%. Results we received by using this setup can be viewed in Tab. 8.9. There we can see that based on training loss best model seems to GRU, however if we look at validation loss we can see that LSTM still have slight edge over both other models. So we can conclude that even though differences in performance became less pronounce as we increase model size LSTM model still maintain slight advantage compare to simpler models.

Model	Train loss	Validation loss
Elman RNN	0.6107	0.6404
GRU	0.5467	0.6329
LSTM	0.5780	0.6268

Table 8.9: Comparison of RNN models with 12 layers of width 784 and 20% dropout rate.

Based on results of these two tested configurations we decided to use LSTM model for further testing.

8.3.1 LSTM

For LSTM we were interested to find best combination of number and size of hidden LSTM layers. Additionally we wanted to know whether adjusting dropout rate would improve model.

As for depth we choose three values for initial testing, those were 3, 6 and 12, to have relatively shallow model, slightly deeper one and comparably significantly deeper one. For the width of these layers we choose 196 which is size embedding and then double and quadruple that size so 392 and 784. In both cases we were interested to see if increasing size of model in their corresponding dimension would have bring sizable improvement in model output quality or not. These two hyperparameters gave us 9 combination we tested, in each test we set dropout rate to 20%, testing of different dropout rates was then afterwards on best model from this testing.

Number of layers	Width of layer	Train loss	Validation loss
3	196	0.4921	0.6389
	392	0.4472	0.6517
	784	0.4114	0.6535
6	196	0.5156	0.6326
	392	0.4786	0.6452
	784	0.4285	0.6493
12	196	0.5983	0.6278
	392	0.5822	0.6292
	784	0.5780	0.6268

Table 8.10: Test and valuation loss for different number of layers and width of layer in LSTM model.

Results of these tests are in Tab. 8.10 where we can see that from perspective of width of a model, we can see that shallower model with 3 and 6 layers show decrease in training with as width increase, however, at the same time we increase in validation loss indicating potential overtraining of wider models. In case of deep models with 12 layers, we see slight decrease in training loss, however this time validation loss stays mostly same for all widths of a model.

When we look at performance of models from perspective of different depths we can see similar behavior indifferently of width of model. We can see by deepening model training loss increase which is counter-intuitive and might indicate issues during training like insufficient regularization, vanishing gradient or wrong learning rate. On the other hand we can slight decrease in validation loss indicating that deeper more were able to generalize information.

For testing of different dropout rate we choose deep model with 12 LSTM layers as these models outperform shallower model significantly based on validation loss, and with each layer having width of 196 as these models showed slightly better results in shallower model while having comparable results to wider model in deep model.

Dropout rate	Train loss	Validation loss
10%	0.5936	0.6351
15%	0.5964	0.6362
20%	0.5983	0.6278
25%	0.5915	0.6342
30%	0.6257	0.6550

Table 8.11: Test and valuation loss for different dropout rate in LSTM model with 12 layers of width 196.

In Tab. 8.11 we see that optimal dropout rate seems to be 20% which resulted in validation loss of 0.6278. This model we consider as the best LSTM model we have.

8.3.2 Decoder-only Transformer

In case of Transformer model when we were trying to find most suitable model we focused on three hyperparameters, two directly influencing model, those were number of transformer layers and number of heads, and one influencing training, which was dropout rate.

Settling of hyperparameters was very similar to LSTM, so firstly we tried to assess best values for first two hyperparameters that influence model structure with dropout rate set to 20% and once we got best results than we test couple of dropout rate values on that specific model. For number of layers we tested same depths of model as in

LSTM meaning shallow model with only 3 layers, bit deeper model with 6 layers and relatively deep model with 12 layers to assess whether deepening model have positive effect on results. In case of number of head hyperparameter we are constricting by the fact that this number have to be divisor of number of dimension on input embedding since model split this embedding equally into the heads. Since our embedding has 196 dimension it's prime factorization is $2^2 \cdot 7^2$. Based on this we choose three values to test, those values were 7, 14 and 49 to see if model would profit more from bigger size of a head or from higher number of heads. These two hyperparameters values gave us 9 combination to test.

Number of layers	Number of heads	Train loss	Validation loss
3	7	0.5057	0.6276
	14	0.5035	0.6266
	49	0.5103	0.6297
6	7	0.4737	0.6416
	14	0.4714	0.6398
	49	0.4716	0.6439
12	7	0.4624	0.6537
	14	0.4301	0.6704
	49	0.4119	0.6729

Table 8.12: Test and valuation loss for different number of layers and number of heads in Transformer model.

If we look at performance of models shown in Tab. 8.12 we see that from perspective of head counts less deep models with 3 and 6 layers have negligible differences with the best value being 14 head in both cases which might be just coincidence since differences are so small. In case of deeper model with 12 layers we see that increasing number of heads cause decrease in train loss so model fit training data better but increase in validation loss so model is worse in generalization of data.

From perspective of depth of model we can interesting see completely opposite behavior to what we saw in LSTM model, where in this case with increased depth of model training loss decreases significantly while validation loss increases, this can be most likely caused by overtraining in deep models.

Since deeper models showed signs of overtraining we decided that for testing of dropout rate we would chose from shallow models with 3 layers, and more specifically we chose model with best both training and validation loss which have 14 heads.

Dropout rate	Train loss	Validation loss
10%	0.4840	0.6433
15%	0.4889	0.6372
20%	0.5035	0.6266
25%	0.5105	0.6232
30%	0.5200	0.6212
35%	0.5225	0.6214
40%	0.5172	0.6226

Table 8.13: Test and valuation loss for different dropout rate in Transformer model with 3 layers and 14 heads.

In results of dropout rate training shown in Tab. 8.13 we can see increasing dropout rate beyond 20% has tangible improvement validation loss while training loss increases but again only to certain point where we can see no measurable improvement beyond 30%. Based on these results we decided to choose as the best Decoder-only Transformer model one with 3 layers, 14 heads and 30% dropout rate, which boast validation loss of 0.6212.

With this we ended with two model to chose from, LSTM model with validation loss 0.6278 and Decoder-only Transformer with validation loss 0.6212. Out of these two we have chosen later one because it not only had slightly lower validation loss but also lower training loss of 0.5200 compared to training loss 0.5983 of LSTM model.

Chapter 9

Results

In this chapter we will report and discuss training and testing of models we chose as best performing on training subset of data in Sec. 8.2 and Sec. 8.3 trained on complete training dataset.

9.1 Training of final models

Both MLP and Decoder-only transformer models were trained on dataset consisting of 156 020 patients around 90% of all patient in our complete dataset. Each of model was trained for 10 epochs.

In case of MLP model we finally settled on batch size of 512 and initial learning rate of 0.0004. After final epoch model finished with training loss 0.0303 and training accuracy of 80.2% meaning there is slight improvement over model trained on only subset of data, but improvement is so small that it seems like we either reached upper limit of what information can be extracted from data or there is unknown flaw in our approach.

When it comes to Decoder-only Transformer model we used lookback value of 20 meaning model always saw last 20 patient records with main goal being to predict 21-st record, since this significantly increased size of single model input we used batch size significantly smaller compared to MLP model, specifically we used batch size of 64. Initial learning rate was set to 0.0025. Training model with these parameters resulted in final training loss being 0.4005 so significantly lower compared to training loss 0.5200 of model trained on subset of data.

9.2 Validation of trained models

Validation of models was done with dataset which contained records of 16 335 patients so more than 9% of patients we have.

In case of MLP model for cost prediction we received validation loss of 0.0305 and accuracy of 80.1%. Meaning both loss and accuracy improved compared to model trained on subset of data with loss of 0.0349 and 78% accuracy, however differences are relatively small. Also validation loss and accuracy is on par with training one so we can sure that model do not suffer from overtraining. In our case we ultimately won't use cost categories of predicted records as their are but we will aggregate them into single category for patient so if from time model would predict category which is off by one in either direction final results would be affected only slightly since there is high possibility that these errors might offset each other. So if we look at percentage of cases when model predicted either precise category or one off we get adjusted accuracy of 96.8% (if we allow model to be off by 2 categories we get accuracy of 99.2%). Another interesting statistics to check is accuracy of model in each cost category to make that can learned to predict correctly in each category not only in those into which majority of data belongs to. We can see results of this check can be seen in Tab. 9.1 where we see that accuracy of decline with decline of their frequency in dataset, however we can see that despite low frequency of categories 5 to 7 model still has over 50% accuracy and only two caterogies under 50% are 8 and 9 which only form very small part of dataset (less than 0.4%).

As for Decoder-only Transformer model trained to predict future records, final validation error was 0.4740 which is much better than validation loss 0.6212 of same model trained on subset of data and is even better or on par with training error of all other different model setups tried during hyperparameter testing.

9.3 Testing of patient future cost prediction

Last 1 000 patients which were not used in neither training or validation were used to test if combination of trained model can succeed in our main task being to predict what would be cost of patient in next year.

Since cost for whole year can be significantly larger that cost for single record we decided to adjust intervals for categories accordingly. Intervals for patient cost can be seen in Tab. 9.2, in this we decided to go with approach where thresholds of next category is about double of previous one.

Category	Dataset frequency	Validation accuracy
1	34.7%	87.5%
2	37.6%	84.4%
3	15.0%	62.5%
4	7.0%	54.8%
5	3.3%	53.7%
6	1.1%	50.8%
7	0.9%	58.1%
8	0.2%	35.3%
9	0.2%	38.2%

Table 9.1: Accuracies of record cost prediction model in each possible prediction category.

Category	Interval
1	[0,100)
2	[100,200)
3	[200,500)
4	[500,1000)
5	[1000,2000)
6	[2000,5000)
7	[5000,10000)
8	[10000,20000)
9	[20000,∞)

Table 9.2: Intervals of patient cost for each category.

Under normal circumstances workflow for computing patient future cost would look like this:

1. load patient data
2. embed patient records
3. compute number of future records
4. predict given number of future records
5. predict cost of future records
6. transform sum of cost of future records into cost category of patient future

However since in this case we want test our model, we modify this workflow slightly. After loading patients data, records from last year are separated from rest and would not seen by model, using remaining records model predict cost based on set workflow, afterwards cost is extracted from separated records and real cost category of patient for that year and is compared to results from model to assess if model was correct or at east close.

Conclusion

REFERENCE SHOWCASE: 3

Bibliography

[1] Anatomical Therapeutic Chemical (ATC) Classification — who.int. https://www.who.int/tools/atc-ddd-toolkit/atc-classification. [Accessed 25-09-2024].

[2] GitHub - essential-data/word2vec-sk: Vector representations of Slovak words trained using word2vec — github.com. https://github.com/essential-data/word2vec-sk. [Accessed 04-03-2025].

[3] GitHub - essential-data/word2vec-sk: Vector representations of Slovak words trained using word2vec — github.com. https://github.com/essential-data/word2vec-sk. [Accessed 19-10-2024].

[4] Google | LaBSE | Kaggle — kaggle.com. https://www.kaggle.com/models/google/labse/tensorFlow2/labse/1?tfhub-redirect=true. [Accessed 18-10-2024].

[5] LSTM — PyTorch 2.6 documentation — pytorch.org. https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html. [Accessed 23-03-2025].

[6] Medzinárodná klasifikácia chorôb - MKCH-10 — nczisk.sk. https://www.nczisk.sk/Standardy-v-zdravotnictve/Pages/Medzinarodna-klasifikacia-chorob-MKCH-10.aspx. [Accessed 16-09-2024].

[7] sentence-transformers/LaBSE · Hugging Face — huggingface.co. https://huggingface.co/sentence-transformers/LaBSE. [Accessed 19-10-2024].

[8] Sivanand Achanta, Rambabu Banoth, Ayushi Pandey, Anandaswarup Vadapalli, and Suryakanth V Gangashetty. Contextual representation using recurrent neural network hidden state for statistical parametric speech synthesis. In *SSW*, pages 172–177, 2016.

[9] Adrien Barbaresi. Simplemma, January 2023.

[10] Andreas Berzel, Gillian Z Heller, and Walter Zucchini. Estimating the number of visits to the doctor. *Australian & New Zealand Journal of Statistics*, 48(2):213–224, 2006.

[11] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[12] Vicent Caballer-Tarazona, Natividad Guadalajara-Olmeda, and David Vivas-Consuelo. Predicting healthcare expenditure by multimorbidity groups. *Health Policy*, 123(4):427–434, 2019.

[13] CDC. ICD-10-CM — cdc.gov. `https://www.cdc.gov/nchs/icd/icd-10-cm/index.html`. [Accessed 16-09-2024].

[14] Yuriy Chechulin, Amir Nazerian, Saad Rais, and Kamil Malikov. Predicting patients with high risk of becoming high-cost healthcare users in ontario (canada). *Healthcare Policy*, 9(3):68, 2014.

[15] Edward Choi, Cao Xiao, Walter Stewart, and Jimeng Sun. Mime: Multilevel medical embedding of electronic health records for predictive healthcare. *Advances in neural information processing systems*, 31, 2018.

[16] Alexis Conneau and Guillaume Lample. Cross-lingual language model pretraining. *Advances in neural information processing systems*, 32, 2019.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[19] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[20] Fangxiaoyu Feng, Yinfei Yang, Daniel Cer, Naveen Arivazhagan, and Wei Wang. Language-agnostic bert sentence embedding. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 878–891, 2022.

[21] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[22] Miao Jin, Qinzhuo Liao, Shirish Patil, Abdulazeez Abdulraheem, Dhafer Al-Shehri, and Guenther Glatz. Hyperparameter tuning of artificial neural networks for well production estimation considering the uncertainty in initialized parameters. *ACS omega*, 7(28):24145–24156, 2022.

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

[24] Waldemar Lopez. *VECTOR REPRESENTATION OF INTERNET DOMAIN NAMES USING WORD EMBEDDING TECHNIQUES*. PhD thesis, 11 2019.

[25] Elizabeth C Lorenzi, Stephanie L Brown, Zhifei Sun, and Katherine Heller. Predictive hierarchical clustering: Learning clusters of cpt codes for improving surgical outcomes. In *Machine Learning for Healthcare Conference*, pages 231–242. PMLR, 2017.

[26] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[27] Nada Mimouni, Jean-Claude Moissinac, and Anh Tuan Vu. Domain specific knowledge graph embedding for analogical link discovery. 06 2020.

[28] Riccardo Miotto, Li Li, and Joel T Dudley. Deep learning to predict patient future diseases from the electronic health records. In *Advances in Information Retrieval: 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20–23, 2016. Proceedings 38*, pages 768–774. Springer, 2016.

[29] Mohammad Amin Morid, Kensaku Kawamoto, Travis Ault, Josette Dorius, and Samir Abdelrahman. Supervised learning methods for predicting healthcare costs: systematic literature review and empirical evaluation. In *AMIA annual symposium proceedings*, volume 2017, page 1312, 2018.

[30] Mohammad Amin Morid, Olivia R Liu Sheng, Kensaku Kawamoto, Travis Ault, Josette Dorius, and Samir Abdelrahman. Healthcare cost prediction: Leveraging fine-grain temporal patterns. *Journal of biomedical informatics*, 91:103113, 2019.

[31] Online. In: Chatgpt vezia 4. *Available at: OpenAI, URL*, Task:.

[32] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

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

[35] Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

[36] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[37] Yi Sun, Yu Zheng, Chao Hao, and Hangping Qiu. Nsp-bert: A prompt-based few-shot learner through an original pre-training task–next sentence prediction. *arXiv preprint arXiv:2109.03564*, 2021.

[38] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.