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

ZADANIE ZÁVEREČNEJ PRÁCE

Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	slovenský
Sekundárny jazyk:	anglický

Názov: Predikcia zhoršenia zdravotného stavu
Prediction of Health Status Deterioration

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
Konzultant:	Ing. Lukáš Palaj
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:

prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.... 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

WRITE ACKNOWLEDGMENT

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: TODO

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: TODO

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

Introduction

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 detection of fraudulent activities, locate contagious disease outbreaks or allocate a future supply of purchased medication.

Another interesting application is predicting a patient's future health, allowing for forecasts of potential diseases they may develop and the treatments they might require. Unfortunately because of large amount of significant factor which information medical records does not contain this task is almost impossible to do.

That's why in this study we will focus on simpler problem and that is to predict expected cost of patient in next year, meaning expected total cost of medications and medical procedures provided to patient during a year. In general we had two tasks to do, firstly embed patients records into numeric vector and secondly to train model to predict future cost of patient based on theirs previous records.

In first chapter we will take a quick look at studies solving similar problems. Second chapter is then dedicated to introduction to data we used. After that in third chapter we introduce models and techniques we used for embedding of records and for prediction task. In forth chapter is short chapter about more technical aspect of software design meaning what language and libraries we used. After than in fifth chapter we specify how we implemented solutions to our two tasks. Sixth chapter is dedicated to preliminary research, meaning to testing of embedding and different model parameters. Finally in seventh chapter we discuss results of final model.

Chapter 1

Similar studies

Necessary prerequisite task for prediction of patient future is to embed patient records into numeric, this task is relatively straightforward for attributes that are already numeric like age, or for attributes which has structured codes like diagnosis however in case of medical procedure we encounter a problem that codes associated to procedures don't have hierarchical structure which means that two similar procedures might have completely different code so we had to come up with way to embed them in a way that similar procedures would get similar embedding, one way to do it is to group medical procedures into clusters and give similar embedding to procedures within cluster and dissimilar one to procedures in different clusters.

For such a task Lorenzi et al. from Duke University in Durham developed novel algorithm called Predictive Hierarchical Clustering [?]. This algorithm was developed for agglomerative clustering of surgical CPT codes. This algorithm uses one-pass bottom-up approach where they utilize EHR, more precisely using 317 predictors, like lab values and patients history, excluding CPT information for 3 723 252 patients and 3 132 CPT codes where each patient have one main surgical CPT code. For each CPT code they create tree containing patients with corresponding code. Afterwards at each iteration, the algorithm considers merging all pairs of existing trees. To compare two trees they utilize two hypothesis, first hypothesis say that data in both trees are generated from same model, while second say data in each tree is generated from models with different parameters. Final value is weighted average of probabilities of these two hypothesis considering data in trees, where weight is probability of first hypothesis 1.1.

$$p(D_k|T_k) = p(H_1^k)p(D_k|H_1^k) + (1 - p(H_1^k))p(D_i|T_i)p(D_j|T_j) \tag{1.1}$$

Where D_k is set of data in merged tree (merged T_i and T_j), T_k is merged tree, H_1^k is first hypothesis, D_i and D_j are data in trees T_i and T_j. During experimental testing they compared result of their approach to clustering CPT codes into 16 clinical groups, to evaluate results they tried to use it predict additional procedures for patients in validation group using their clustering and clinical clustering, then compute area under receiver operator curve (AUROC) and area under precision recall curve (AUROC). They found few percentage improvement in these statistics using their model compared to clinical clustering.

In the end we decided to not try this approach in our work since it is not really able to distinguish whether two procedures are similar considered similar because of their real similarity or only because they commonly appear together as a determination or treatment to specific diagnosis. This distinguishment is important for since despite procedures which similarity is only that they appear together might have vastly different cost associated to them which could cause issue for prediction model.

Main goal of our study is to predict cost of patient in the future, so how much money would be spent on drugs and procedures for specific patient.

One of similar studies in this regard is study by Caballer-Tarazona et al. [?] in which they tried to predict future cost of the patient primarily using what they called "Aggregated Clinical Risk Group 3" computed from standardized Clinical Risk Group (CRG), this variable consist of two parts, first in one of nine grouped CRGs and second part is one of six levels of severity. They also tested adding additional information such age or sex of patient to see if it helps improve model. To avoid issues with a possible large mass of observations with zero-cost they developed two-part model which first part uses logit model to determine probability of cost greater than zero and after that second model predicts expected cost in case of positive cost, for this second part they tried log-linear ordinary least square model and generalized linear model with log link. Final prediction was then multiplication of probability of non-zero cost and expected positive cost. Their models managed to get adjusted R^2 values of $46.4 - 49.4\%$, which they comment as comparable to those obtained in other similar studies that used different patient classification systems.

This approach was in the end not viable for us since it utilize specific attributes like CRG and severity level of patient which we did not had.

Another study focused on future patient cost predictions is study by Mohammad Amin Morid at al. [?] which focuses on comparison of multiple model to compute future cost category of patient using medical claims and pharmacy claims data.

Their data for single patient consisted of 21 features all connected to amounts spend on that patient, more specifically features of their data point were for example total cost for patient, total cost for patient in last 6 months, number of months with cost above patient average or linear trend of cost.

They tried to predict to which of 5 cost categories would belong in the future, in order to do that they tried and compared multiple prediction models such as Linear Regression, Decision Tree and Artificial Neural Network (ANN). In case of Linear Regression they tried also versions utilizing regularization like Lasso or Ridge models and in case of Decision Tree models they tried also improved models based on it like Random Forest, Bagging or Gradient Boosting techniques.

In their testing most successful model ended up being Gradient Boosting which were able to predict correct cost category in almost 95% of cases, however in highest cost bucket which should have contained only 2% of population model was successful only in 37.5% of cases. Second and third best models were ANN and Ridge model, which both have very similar results with around 91.5% total correct category assessment and over 50% correct category assessment in each cost category. So in conclusion they assess that Gradient Boosting provides the best cost on cost prediction models in general, with ANN providing superior performance for higher cost patients.

This study gave us hope that our data might contain necessary information for prediction as most of features in their study and computed using information of cost of drug or procedure which is present for each record in our dataset while also containing much more about cause of that cost.

[?]

Chapter 2

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 model we used completely artificial data with matching structure to data in NCZI database. The data were generated in a manner that simulates realistic patient records.

Data we have can be split into two categories, those are patients records and mapping table.

2.1 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 about 173355 patients. Each patient have at least 70 records.

2.1.1 Records of medical procedures from ambulatory health care

Each row of this dataset is equivalent to single patient record, meaning single medical procedure done to them, in total this dataset consist of (ADD NUMBER HERE) records. Each record consist of these variable:

- 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 we use most of these information, date of the procedure combined with patient age is used to create timestamp information used to order all records for patient as well as one of the dimension of record embedding. Identification code of patient is used to be able to gather all records for single patient. Identification codes for diagnosis and procedure are matched with their corresponding numerical vector and embedded into vector corresponding to record (see 4.1). Cost is encoded into cost category and used as information of cost associated with embedding of record.

2.1.2 Records of prescribed medicines

Similarly to dataset containing procedures, each row is equivalent to single patient record, in this case it's single drug prescription for that specific patient, in total we have (ADD NUMBER HERE) records of prescribed drugs. Each record consist of these variable:

- 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

Use of these variable is also similar to procedures, with only difference that instead of using encoding procedure into record embedding we encode drug.

2.2 Mappings

Other than datasets containing patients data we use three mapping files to map identification codes used for attributes in patients records datasets to corresponding embedding vectors. Mapping datasets are:

- Diagnosis mapping
- Drug mapping
- Medical procedure mapping

These datasets are publicly available dimensional table from maintained and used by NCZI. In general all three mapping table have comparable structure consisting of three main attributes. First is internal identification number used by NCZI to join these mappings on patients records. Then second attribute is official code, in case of drugs and diagnosis it's an internationally standardized structured code while medical procedures uses code specific for Slovakia which unfortunately lack any kind of meaningful structure. Final last important part is textual description.

We used these dimensional table to generate embedding vectors for drugs, diagnosis and medical procedures. Process of this embedding is explained in section 6.1.

Chapter 3

Theory - models

During our research we utilized multiple different machine learning models, in this chapter we introduce them from more theoretical point of view.

3.1 Multilayer perceptron

Multilayer perceptron is a feed-forward neural network, so network where data flow single direction or in other words neurons don't form cycles. This network consist of fully-connected, sometimes called dense, layers with non-linear activation functions as shown on Fig. 3.1, where we can see that this model can be split into three parts which are input layer which load the data, hidden layer which are trying to extract desired information using linear transformations and activation functions and finally output layer which contains final linear transformation followed by activation to output extracted information. Each layer can be described using this formula shown in Eq. 3.1, where h_i is resulting vector of i-th layer, $act()$ is a non-linear activation function, W_i is weight matrix of i-th layer, h_{i-1} is resulting vector from previous layer and b_i is bias vector.

$$h_i = act(W_i h_{i-1} + b_i), \tag{3.1}$$

3.2 Recurrent neural network

In general recurrent neural network is type of neural network that in some way uses results from previous step or steps in order to improve prediction. These models are used for ordered data where next step is dependent on more than single previous one.

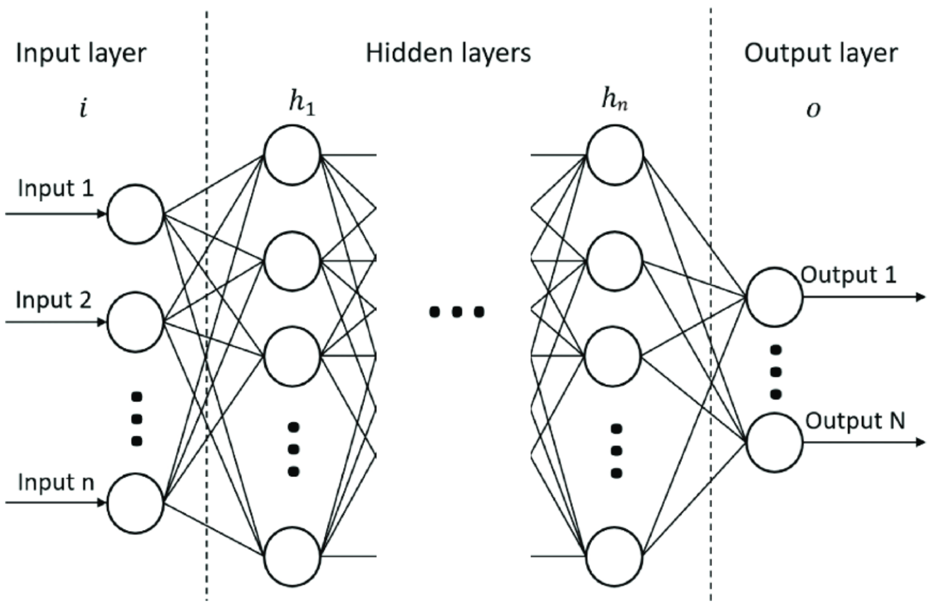


Figure 3.1: Architecture of multilayer perceptron [?].

3.2.1 Elman RNN

Elman RNN also known as simple recurrent network (SNR) is type of recurrent neural network that utilize results of hidden layer before activation in previous step as an additional input in next one. We can see this architecture on figure 3.2, where on left we can see how forward propagation look and on right how it looked unrolled over time, we see that middle layer which is a hidden layer of model gets two inputs, first is input vector x_t, where t denotes step (usually time step), which is multiplied with matrix of weights W_i and second one is vector h_{t-1} which is result of this layer also called context vector in previous step, which is multiplied by different matrix of weight denoted as W, these two results after they are then summed together and result goes to activation layer which create resulting new vector h_t [?], this whole process is summarized in Eq. 3.2(where b_i and b are bias vectors).

$$h_t = act(x_t W_i^T + b_i + h_{t-1} W^T + b). \tag{3.2}$$

Usually this result is directly used as output or go into single fully-connected feed-forward layer. In multi-layered version of this network, result of first hidden layer would go into next hidden layer together with result of that specific layer in previous step and again both would get multiplied by matrices of weight specific for that layer, summed and results goes through activation function.

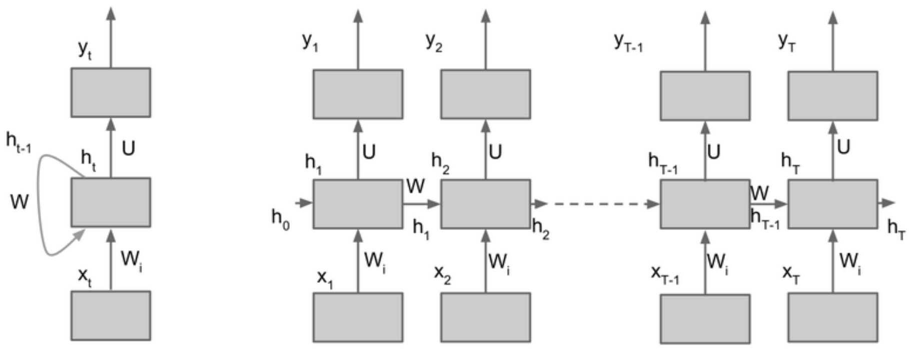


Figure 3.2: Architecture of Elman RNN [?].

3.2.2 Gated Recurrent Unit

Gated recurrent unit is more complex RNN compared to Elman RNN. This model was developed as simplification of even more complex LSTM model. We can say that it consist mainly of three parts that interact with each other and together create final prediction. Those parts are reset gate, update gate and candidate hidden state computation. We can see structure of hidden layer of GRU on figure 3.3. On this diagram sigmoid means fully-connected feed-forward layer with sigmoid activation function and tanh means fully-connected feed-forward layer with hyperbolic tangent activation function.

Reset gate is on left side of diagram, it's task is to using input and previous hidden state vectors modify previous hidden state vector which goes into candidate hidden state computation. This should helps capture short-term dependencies in time series by removing part of information from previous hidden state vector.

Candidate hidden vector computation is part that is similar to Elman RNN with two differences, firstly inputted hidden layer is modified by reset gate result and secondly resulting vector from this part is only candidate which goes into further calculation. This candidate contains mostly current and short-term past information thanks to specific vectors that input consist of.

Update gate, similarly to reset gate, uses concatenation of input and previous hidden state vector on input, but this time results are used to modify both previous hidden state and candidate hidden state in a way that resulting hidden state is weighted average of those two vectors where weights are results from this gate. This should help to capture long-term dependencies in time series from previous hidden state vector and reintroducing it into final hidden state by combining it with candidate hidden state vector that should contain mostly current and short-term information. This whole architecture is shown on Fig. 3.3.

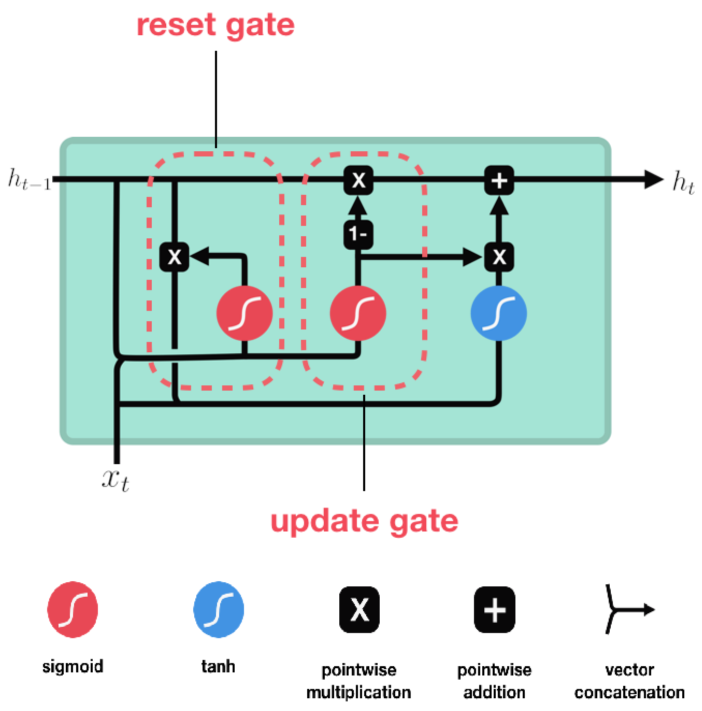


Figure 3.3: Architecture of GRU hidden layer.

Multi-layered version is achieved by stacking hidden layers where hidden state vector of one layer is input vector of next one.

3.2.3 Long Short-Term Memory

As mentioned earlier, LSTM is more complex predecessor of GRU. This model was developed with aim to mitigate vanishing gradient problem, that cause lack of long-term memory in models like Elman RNN, mainly by introducing memory vector, sometimes called memory cell, that should maintain information over longer period. As we can see in figure 3.4 this model consist of three gates, candidate memory computation. Legend in this diagram have same meaning as one in GRU architecture diagram.

All three gates and candidate memory computation has same input, which consist of input and previous hidden state vectors. In all of them this input goes into fully-connected feed-forward layer and then into activation function. In case of gates this activation function is sigmoid function, that bound all values into range (0,1), while candidate memory calculation uses hyperbolic tangent as activation function.

Forget gate has the task of removing unnecessary information from memory vector by elementwise multiplication of it's result with previous memory vector.

Input gate together with candidate memory vector computation are meant to introduce new information into memory cell after adjustment from forget gate. Firstly model computes candidate memory then, then this vector is adjusted by input gate result and finally added to the modified previous memory vector. This results in new memory vector.

Finally output gate is used to determine how much each part of memory should contribute into new hidden state vector. We can see diagram of this process on Fig. 3.4.

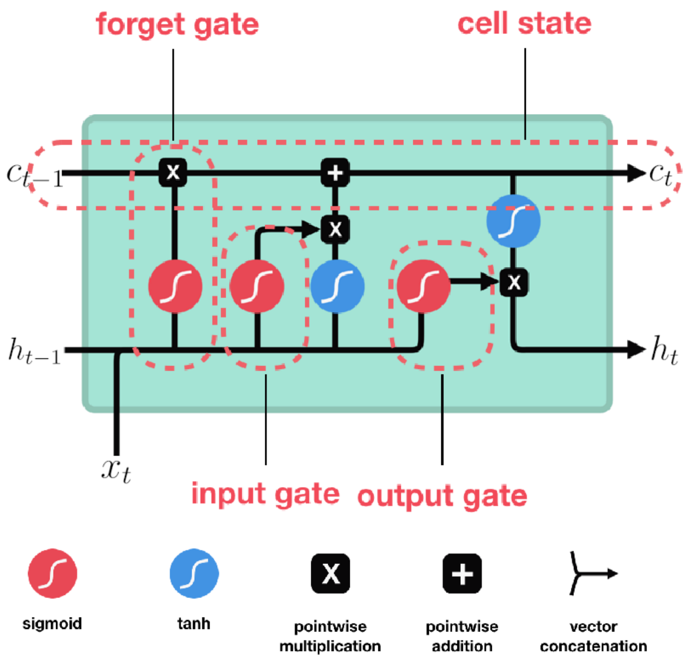


Figure 3.4: Architecture of LSTM hidden layer.

3.3 Transformer

Transformer is a deep learning model architecture introduced in 2017 by Vaswani et al. [?] that's especially good at handling sequences, like text. This architecture consist of encoder and decoder as shown in Fig. 3.5. Encoder in this model is meant to contextualized representation of input while decoder takes this contextualized representation and mixes together with previous outputs to get prediction of next output. Other than these two main blocks this model usually also contains Tokenizer , Embedding layer and Positional encoding to prepare in going data and Fully-connected Feed-forward layer with softmax activation function to turn result of decoder into probability distribution of possible tokens.

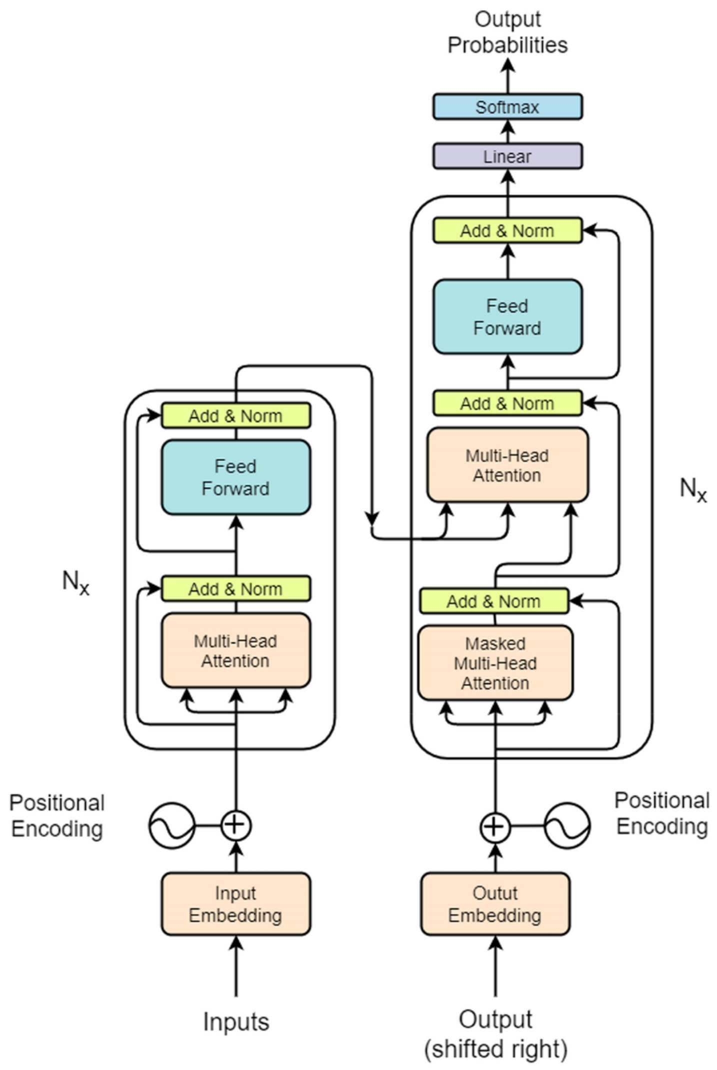


Figure 3.5: Architecture of one encoder-decoder block in transformer model from original "Attention is all you need" paper [?].

3.3.1 Tokenizer, Embedding layer and Positional encoding

First part of Transformer model is usually input preparation, this part may differ based on type of data used. This part consist of three parts:

- Tokenizer - this layer split input into tokens, for example if input is textual, tokens might words or syllables
- Embedding layer - task of this layer is to transform each one of input tokens into numerical vectors of same length

- Positional encoding - finally last preparation layer add to each embedding of token vector which encodes their position with regards to other tokens

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

3.3.2 Encoder

Encoder architecture consist of multiple attention blocks where each block consist of multi-headed Self-attention and Multi-layered Feed-forward Neural Network, after each of these step embeddings from step input are added to output and result is layer normalized, adding of step input embedding is done to create residual paths that helps mostly with vanishing gradient problem and layer normalization then helps by making sure results neither explode neither go to zero as well as it bring bit of additional non-linearity to model. Firstly input embeddings are split into parts that each goes into separate head Self-attention.

Architecture of Self-attention is shown on Fig. 3.6 where we can see that each input token get multiplied by key, query and value matrices to receive their key, query and value vectors, after that each query is multiplied with each key to create attention matrix which should encode how much each token influence other, after that matrix goes into column wise softmax function to normalize it and finally it's multiplied my matrix of value vectors to create new embedding of tokens that should contain not only original information but also influence information.

Results of each Self-attention head are then concatenated back into new embeddings that have same dimensions as original ones, then after adding residual connection and normalization results goes into Multi-layered Feed-forward Neural Network to further extract information from embeddings.

3.3.3 Decoder

Architecture of decoder, similarly to encoder, consist of multiple attention blocks, however in this case attention block consist of three parts instead of two.

First is masked multi-headed Self-attention which is similar to unmasked Self-attention with only difference being application of masking to the attention matrix before softmax function to make sure that words on some specific positions does not affect words on other specific positions. Decoder use what's called casual or look-ahead

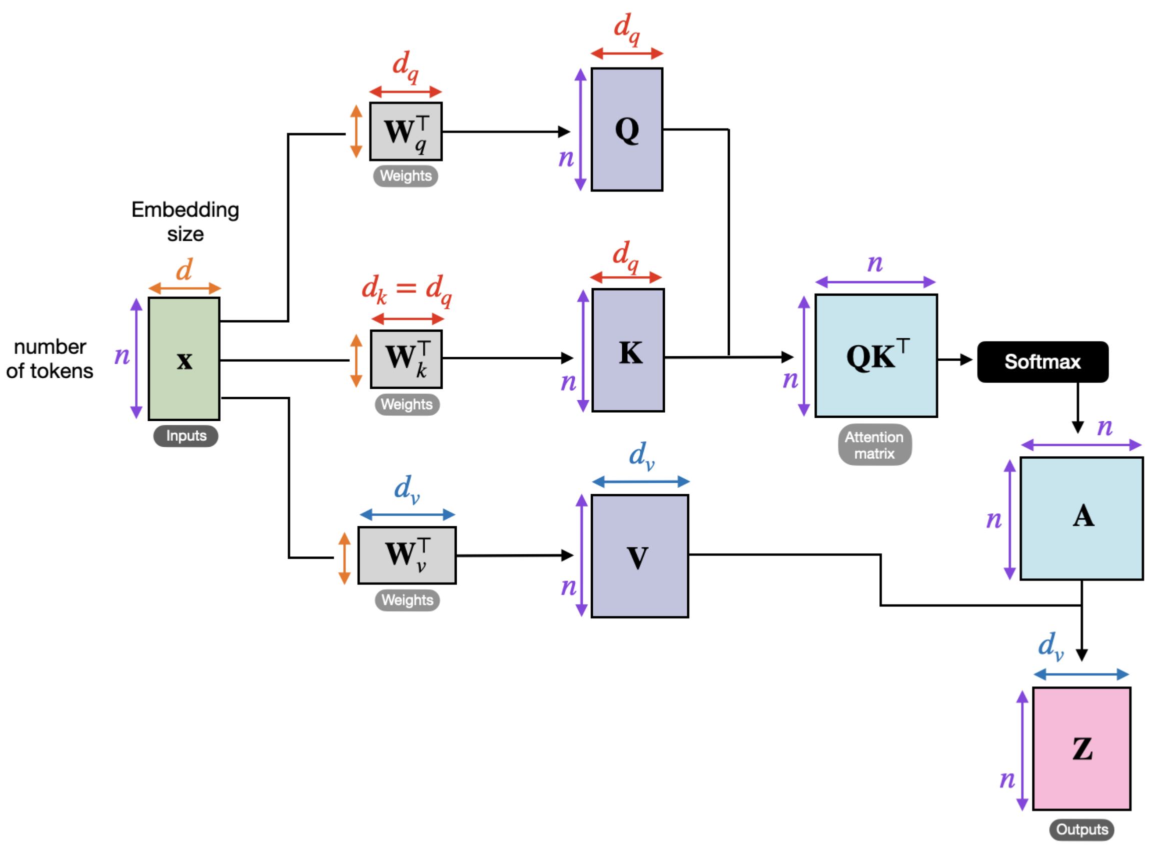


Figure 3.6: Architecture of Self-attention mechanism.

masking which forbid future tokens to affect past ones, in our case this guarantees that record cannot be affected by record which happened after so in the future from it's perspective.

Second part is multi-headed Cross-attention, which takes two lists of token embeddings and compute effect of tokens in first list onto tokens in second list. In this case key and value vectors are computed from contextualized representation computed by encoder while query vectors are computed from results of self-attention in decoder, other than that remaining architecture is same as unmasked Self-attention.

Final part of decoder attention block is multi-layered feed-forward neural network that further extract information from embeddings.

3.3.4 Fully-connected Feed-forward layer

After that results of decoder goes into final Fully-connected Feed-forward layer which change dimensionality of the decoder output from embedding size into vocabulary and apply softmax activation on result. This way we expect to get in last token probability distribution through all possible tokens. In our case we changed this last part. We run into problem where most of our records (our tokens) are unique creating extremely large vocabulary, this was caused partially by many possible combinations of disease, drug and medical procedure which are encoded in record and partially by timestamp which add additional uniqueness since it's improbable that two patients would get for example same drug, for same disease, at same age. Because of this we decided to remove softmax activation, changing fully-connected feed-forward layer in a way that result maintain dimension of embedding and added function that split result, find closest embedding of each part and concatenate result together leaving us with specific new embedding prediction instead of probability distribution.

3.3.5 Decoder-only Transformer

Decoder-only version of Transformer model is simplified version of model which completely exclude Encoder part of model and from Decoder part it excludes Cross-attention mechanism. This model is usually used if we wanna generate next tokens without having stable context that would normally go into Encoder part.

This means that this simplified architecture is much more similar to standard RNN in terms of input and expected output.

3.4 Language-agnostic BERT Sentence Embedding

Language-agnostic BERT Sentence Embedding model also known under abbreviation LaBSE is model trained with main goal being to generate similar representation to pairs of sentences which have same meaning and are only translations in two different language [**?**].

Architecture of LaBSE model consist of 4 parts [**?**]:

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

3.4.1 Encoder-only Transformer (BERT model)

First and most important part of LaBSE model is transformer, which is a deep learning architecture. More specifically LaBSE uses BERT so bidirectional encoder representations from transformers model which is encoder-only transformer architecture meaning this model does not contain decoder found in standard transformer which is usually used for prediction, because of this BERT model is focused in extracting contextual information from input text. Architecture of standard BERT model looks like this:

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

Tokenizer layer

First layer is tokenizer, this layer takes input text split it into tokens which in case of BERT model is called PieceWise tokenizer which split text into subwords so something close to syllables. PieceWise tokenizer has advantage compared to different tokenizers that use either words or characters. Compared to character wise tokenizing, subwords contain more information than characters, and compared to word tokenizer is that there a lot less subwords than words and are more similar across multiple languages, creating much smaller vocabulary which is especially important for multilingual models. After split this layer assign integer number to each unique token, LaBSE model vocabulary distinguishes around 500 000 different tokens.

Embedding layer

After that comes embedding layer which assign real number vector to each token, more specifically BERT model compute three different embeddings and add them together and normalize result to get final one. First is token type embedding, which is basic embedding where each token in vocabulary is assigned it's unique embedding. Second is positional embedding, as name suggest this embedding contain information about where in the sequence token is found giving additional information. Third and final embedding is segment type, which encodes information about to which segment, usually sentence token belong, important for embedding input text consisting of multiple sentences.

Encoder

Third and most important layer is encoding. This is the layer in which contextual information are mined from the text. Architecture of this layer is same as architecture explained in Sec. 3.3.2. In case of BERT used in LaBSE there are 12 attention blocks.

Task layer

Each head of self-attention layer takes a input set of embeddings and to compute new set of embeddings which should encode not only original information but also information about relationship between original ones, in other words it encodes effect of tokens onto each other. To do that it compute for each input embedding three vectors usually called key, query and value by multiplying input embedding with three matrices which values are learned during training. After that to get new embedding query vector is multiplied with matrix created from key vectors so resulting vector is vector of dot product of single query and all keys, this vector then goes through softmax function to normalize result. In some transformer models before softmax this vector get masked. Masking is done by setting values where key vector belongs to later embedding than query to minus infinity, this way after softmax this values become zeros. This is done so that later embedding does not affect previous ones. It's mostly useful in models trained to predict next token in order for model to learn to predict only based on tokens from past on not future. However in case of model like BERT where emphasis is on extracting as much information from input text masking is not done. Final step to get new embedding is to multiply vector we got after applying softmax with matrix composed of value vector to get linear combination of value vectors. This resulting vector is new embedding. This is done for all query vectors. List list resulting vectors is output embedding of attention head. Since this model use multi-headed attention more specifically in case of LaBSE 12-headed attention, this process is simultaneously independently done 12 times and results of each head are than concatenated into final result of attention. By using multiple head we expect that each head can extract different information from input and final concatenation give us result that contains all extracted information.

This concatenation of new embeddings goes than into multi-layered feed-forward neural network sometimes also called Multilayer Perceptron or MLP for short, architecture of this model is explained in Sec. 3.1.

Result goes to next attention block and process is repeated again. Result of final attention block then goes to last part which is task layer, this layer can be viewed as simple decoder that map resulting embeddings back into token space. Based on this results is then model pre-trained. Training process of BERT model usually consist of two tasks on which are model trained at the same time. First is Masked Language Modeling (MLM) where 15% of input tokens are masked meaning either replaced by mask placeholder or by random different token, after that masked input goes into model, then from resulting tokens are taken those on position of masked tokens in input and are compared to correct token before masking which creates error that is back-propagated through model updating it's parameters [?]. Other task usually used is called Next Sentence Prediction (NSP), in this task model gets input which start with special classify token and after that two spans of texts separated by special separator token. Task of model is to say whether these two spans of text can appear one after another or more precisely whether they appeared one after another in training corpus and put this information into first token of result encoded by two special tokens either "is next" or "not next" token and similarly difference between expected and resulting first token create error that back-propagates through model [?]. However in some BERT-based model like LaBSE second task is replaced with translation language modeling (TLM), this task is extention of MLM in which model gets two concatenated sentence instead of one and where the second sentence is translation of the first in another language, rest of the task is than same as in MLM, so whole input gets masked and model is tasked to predict masked tokens [?]. Task layer is used primarly in only during pre-training and is omitted when model is used for different task as many use-cases does not need tokens in results but use embeddings from encoding layer as form of text encoding which is that further used in task specific layers on which then model is fine-tuned.

3.4.2 Pooling layer

After BERT model returns embeddings of all input tokes, these then needs to be aggregated into single vector which corresponds to embedding of whole input. This aggregation is done using pooling layer. In case of LaBSE this pooling is done simply by taking embedding of first token which is an embedding of special classify token added to beginning of BERT input.

3.4.3 Dense layer

Next layer is standard feed forward dense layer with use hyperbolic tangent activation function. Number of input and output neurons is same, and additional bias neuron is used.

3.4.4 Normalization layer

Final layer is normalization which task is only to normalize resulting vector, so divide vector by number in order to have final vector with L_2 norm equal one.

3.5 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 distance between two embeddings contains some underlying information about those words such as their similarity. There are two main approaches to implementing Word2vec:

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

3.5.1 CBOW approach

Model using CBOW approach gets sequence of words called context with one being missing and on the output it's trying to predict missing target word. Model initially assign one-hot encoding to each word in it's dictionary. During training, each word from context is firstly converted into it's one hot encoded embedding which is then multiplied by weight matrix to get their lower dimensional dense embedding, dense embedding of all words in context are then averaged out and resulting embedding goes into hidden layer which transform vector back into dimension of vocabulary, finally softmax function is applied to get probability of each word from vocabulary to be predicted as missing word. We can see this architecture in 3.7. Training is usually done using fixed context window moving along training text.

3.5.2 Skip-gram approach

Skip-gram approach works in opposite way, where model gets target word and is trying to predict context around it. Similarly to CBOW, input word is firstly one-hot encoded then multiplied by weight matrix to transform it into dense encoding, the goes into next layer to transform it back into dimension of vocabulary goes into softmax creating vector of probabilities of surrounding context words. Error of Skip-gram is the sum of the negative log-likelihood of all context words. We can see this architecture in 3.8.

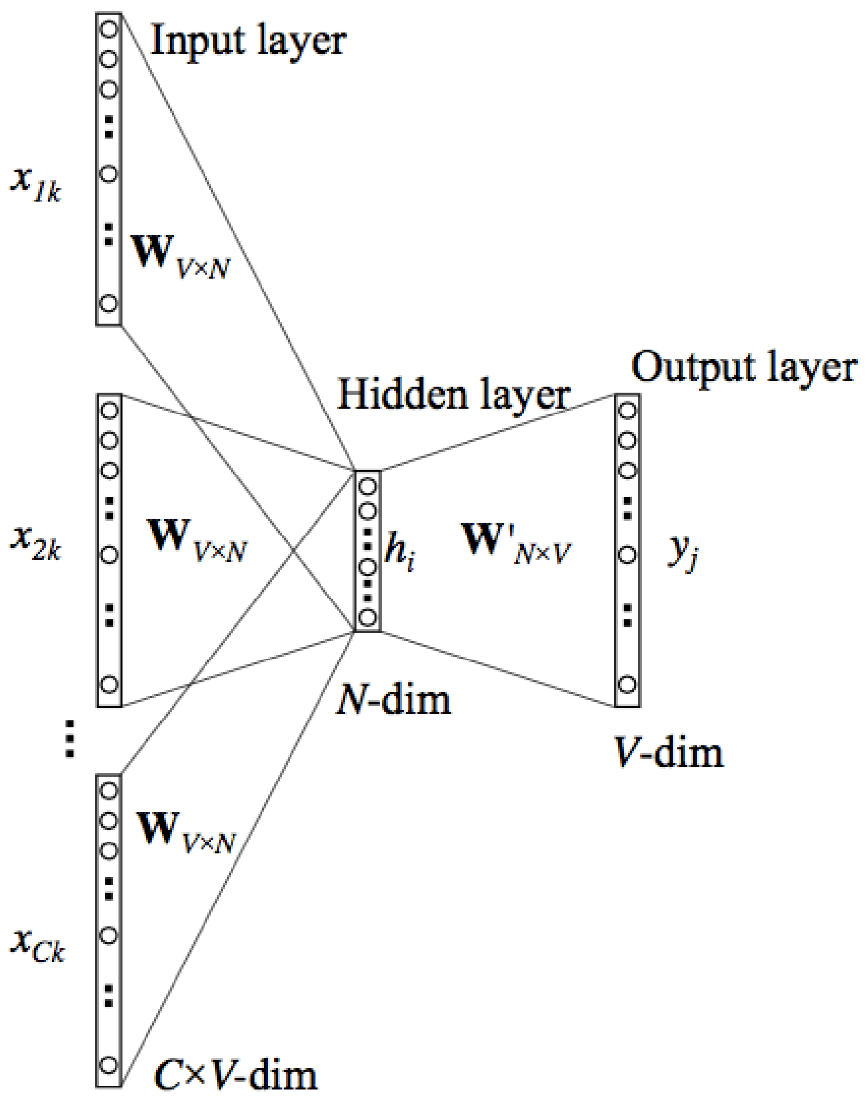


Figure 3.7: Architecture of NN to train CBOW implenetation of Word2vec model [?].

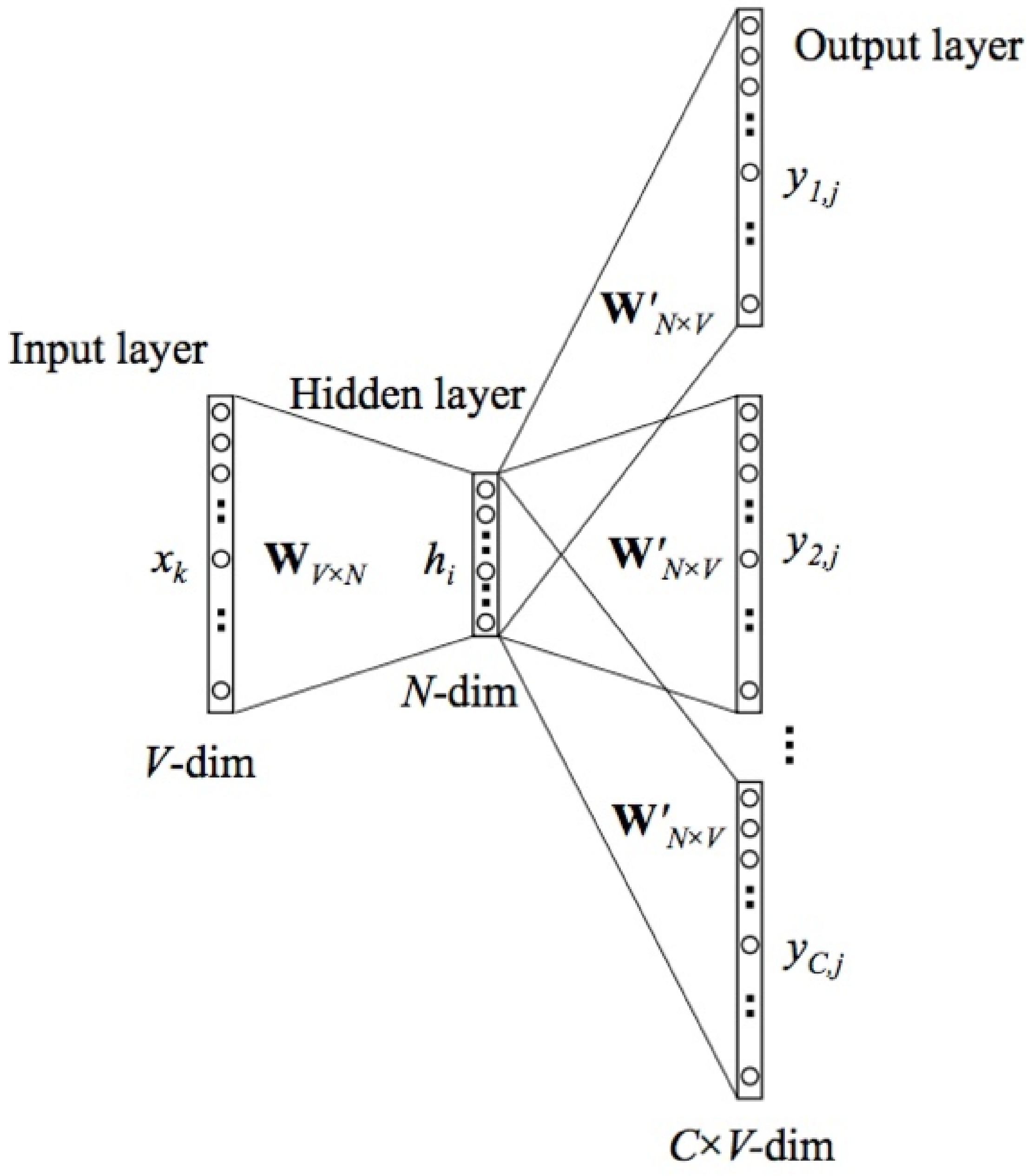


Figure 3.8: Architecture of NN to train Skip-gram implenetation of Word2vec model [?]

Chapter 4

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

4.1 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.

4.1.1 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 6.1.1, 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.

4.1.2 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 [?] most precisely version of this code that is used in Slovakia and is better known by the acronym MKCH-10-SK (Medzinárodná klasifikácia chorôb) [?].

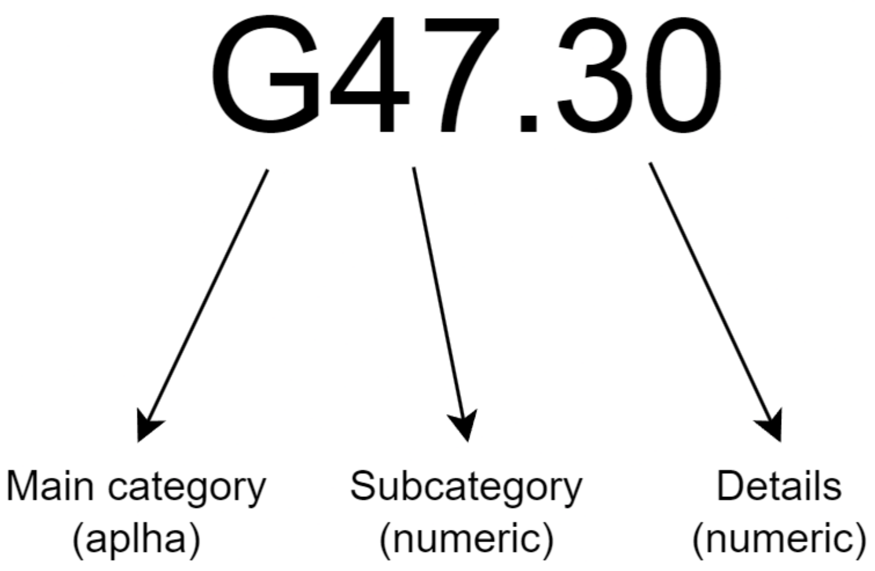


Figure 4.1: Structure of MKCH-10 code.

This code consist of three parts as shown on Fig. 4.1. 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 CITE, however Slovak version MKCH-10 codes contains at most two numeric characters to specify disease cite 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.

4.1.3 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. 4.2. 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 4.1.2 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.

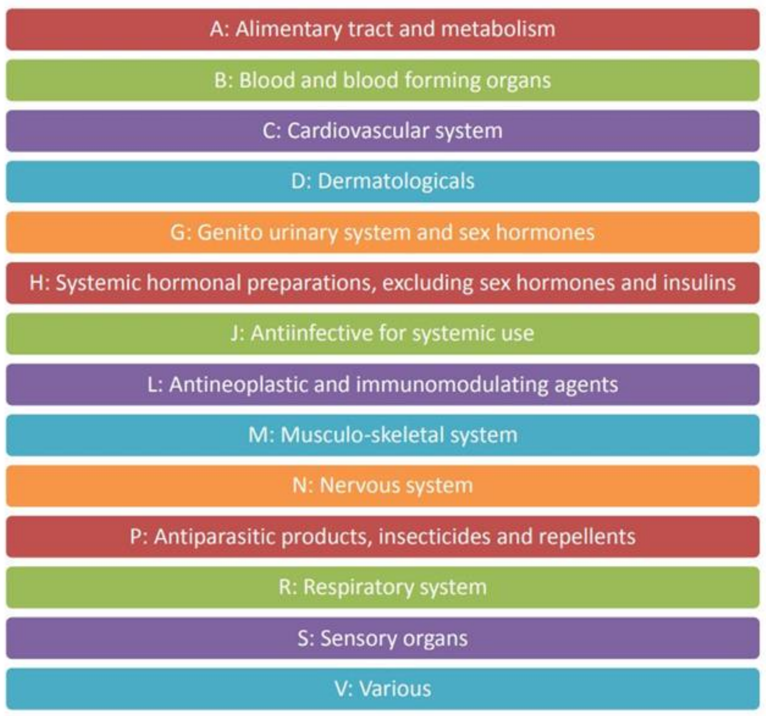

A: Alimentary tract and metabolism
B: Blood and blood forming organs
C: Cardiovascular system
D: Dermatologicals
G: Genito urinary system and sex hormones
H: Systemic hormonal preparations, excluding sex hormones and insulins
J: Antiinfective for systemic use
L: Antineoplastic and immunomodulating agents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

Figure 4.2: Fourteen main anatomical or pharmacological groups and their corresponding first level ATC code [?].

4.1.4 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 3.5, 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 3.4, 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.

4.2 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 [?] 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.

4.3 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. 3.2. Last model we tried was Transformer explained in Sec. 3.3, more specifically we tried Decoder-only Transformer model.

4.4 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. 3.1.

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. [?] which we cover in Chap. 1.

Chapter 5

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.

5.1 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 [?] 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 [?] which we use for computations such as random number generation, computations of mean and standard deviation for data normalization and many more.

5.2 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 [?], 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 [?], 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 [?], 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 [?], 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 [?], which we used to load Word2vec model trained specifically for Slovak language

5.3 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 [?]. 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 6

Implementation

6.1 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.

6.1.1 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.

6.1.2 Diagnosis embedding

As discussed in Sec. 4.1.2 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. 6.1 where each part is highlighted by different color and all values are rounded to two decimal.

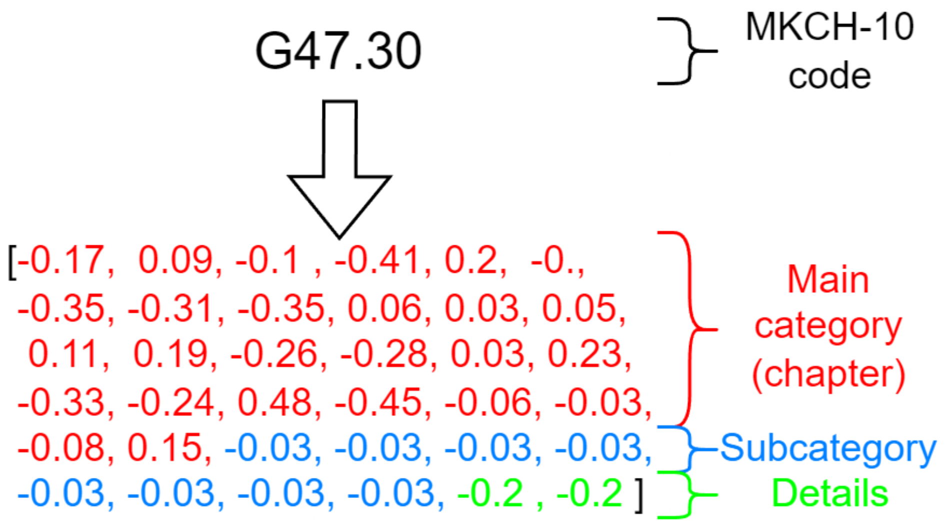


Figure 6.1: Showcase of resulting embedding of specific diagnosis (rounded to two decimal places).

6.1.3 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. 4.1.2 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. 6.1. 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 6.1: 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.

6.1.4 Medical procedure embedding

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

As discussed in 4.1.4, 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 [?]. 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.

6.2 Prediction of record cost

As discussed in 4.4 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. 6.2 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. 6.2.

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

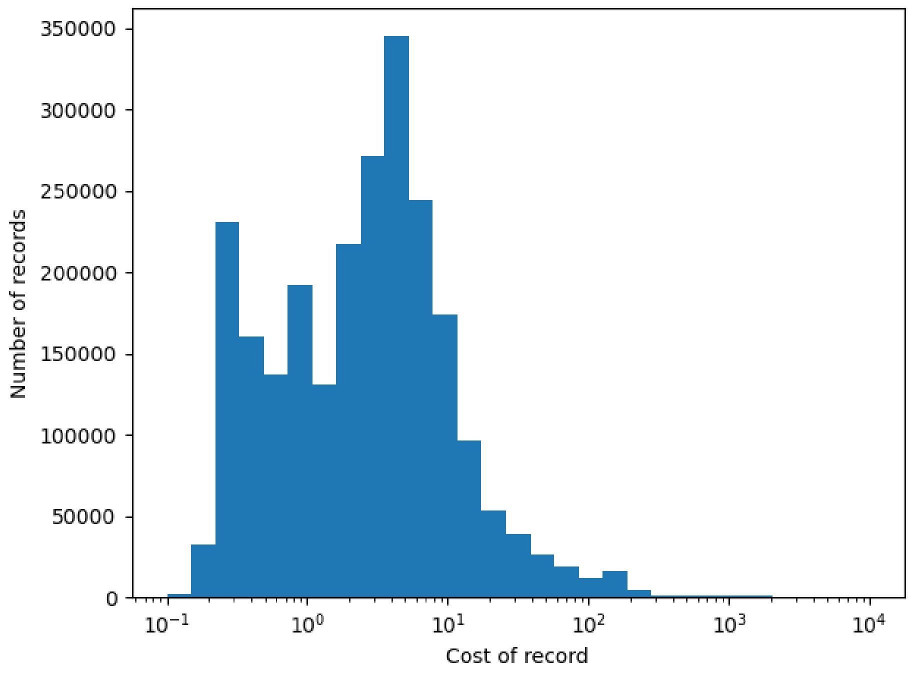


Figure 6.2: Histogram showing distribution of cost of records in training dataset.

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 6.2: Intervals of 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) [?], 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.

6.3 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:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y_i})^2. \tag{6.1}$$

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:

$$SubpartWeightedMSE = \frac{1}{p}\sum_{j=1}^{p}(\frac{1}{l_j}\sum_{i=1}^{l_j}(Y_{s_j+i} - \hat{Y_{s_j+i}})^2). \tag{6.2}$$

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 [**?**], 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 7

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.

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 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. 7.1 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 7.1: 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 7.2: Size of clusters and frequencies of first level diagnosis codes in them.

In Tab. 7.2 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.

7.1.2 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. 7.3. 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 7.3: 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. 4.2 we used K-means algorithm with 14 clusters and got these results:

Based on results we can see on Tab. 7.4 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 7.4: Size of clusters and frequencies of first level drug codes in them.

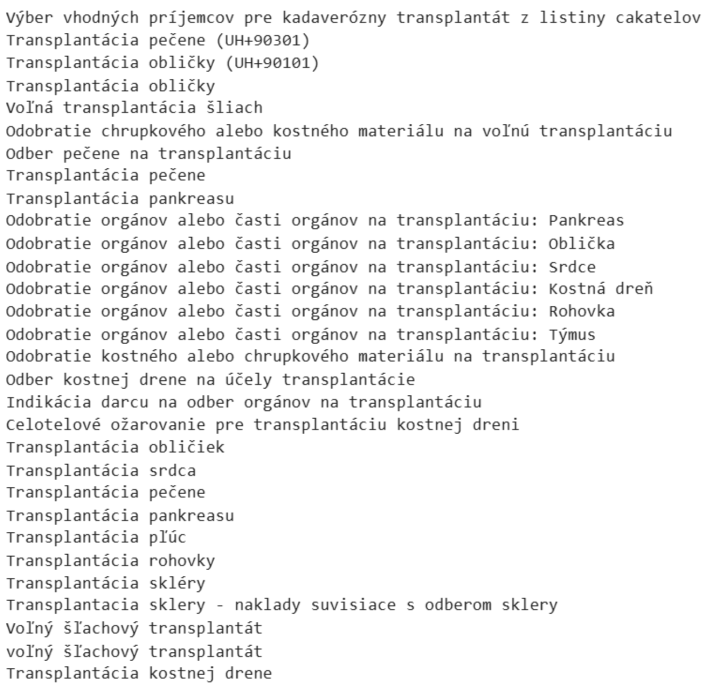
7.1.3 Medical procedure embedding

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 preserved information. After running this algorithm we ended up with 119 dimensional embedding that explained 90.4% of variance of original embedding.

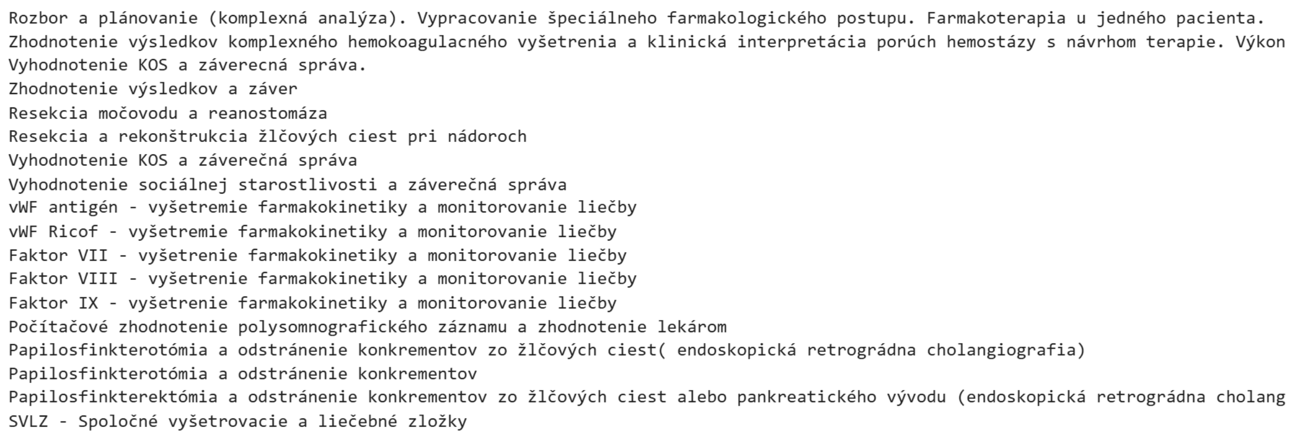
To confirm that resulting embedding successfully encode description of procedure and gave similar procedures similar embedding we did same thing as in case of drug and disease embedding validation and try to group embeddings using K-means algorithm. However in this case we did not have any information 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. 7.1 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. 7.2 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 7.1: 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 7.2: Medical procedures grouped in cluster number 45.

7.2 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. 6.2 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. 7.5.

First loss function we tested was mean square error (MSE) loss, which computes loss based on equation shown in Eq. 7.1 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. 7.5 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.

$$Loss = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{7.1}$$

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. 7.5, 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. 7.6. 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 7.5: 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 7.6: Accuracies of Gradient Boosting models with different maximal depths of tree.

Results of Ridge model testing are shown in Tab. 7.7 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 7.7: 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 for main training.

7.3 Prediction of future records

As said in 6.3 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. 7.8 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 7.8: 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. 7.9. 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 7.9: 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.

7.3.1 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 7.10: Test and valuation loss for different number of layers and width of layer in LSTM model.

Results of these tests are in Tab. 7.10 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 7.11: Test and valuation loss for different dropout rate in LSTM model with 12 layers of width 196.

In Tab. 7.11 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.

7.3.2 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 7.12: 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. 7.12 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 7.13: 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. 7.13 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.

Chapter 8

Results

Conclusion

REFERENCE SHOWCASE: 2