Final Master's Thesis

Master's degree in Automatic Control and Robotics

Automatic translation between layman and HPO terms using machine learning algorithms

Master Thesis

Author: Enrico Manzini

Tutor: Alexandre Perera Lluna

Jon Garrido Aguirre

Call: June 2019

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Abstract

Linguistic differences between specialists and laymen still represent an obstacle for a successfull communication in technical environments. This is especially true in the medical domain where the linguistic gap between clinicians and patients is a considerable issue: from one side, diseases and symptoms must be described with a very specific vocabulary to avoid doubts and ambiguity; from the other side, it can not be expected for patients, that are the main source of information for an accurate diagnosis, to use the same technical jargon as physicians in order to describe their symptomatology.

The main objective of this project is to investigate a possible solution to this issue using a deep learning approach to support the collection and description of all the traits of patients with rare disease in the Share4Rare network. Machine learning techniques will be used to develop a machine translation model that will be able to transform the input layman terms into specific medical concepts.

In order to achieve this objective, the most common deep learning methods used in Natural Language Processing will be explained and analyzed, with a particular focus on word embedding techniques, convolutional neural networks and recurrent neural networks. Then, three models that combine these techniques will be proposed, trying to outline strengths and weaknesses of each one. All the models will be created and tested with Python, a high-level, general-purpose programming language. The neural network architectures will be created using Keras, an open-source deep learning library for Python.

The proposed models will be trained and tested using the lexicon from the Human Phenotype Ontology, a formal ontology of human phenotypes with the aim of becoming the standard vocabulary for clinical databases. Terms in the Human Phenotype Ontology contain synonyms and descriptions of the phenotypes to which they refers that will be used as input for the different models. Results will be evaluated with cross-validation, and domain specific performance metrics will be adopted to carry out a specific analysis of the outcomes.

Contents

List of Figures

List of Tables

Glossary

$D_t(t_i)$ List of Wikipedia pages related to the i-th HPO term. 32

$R_t(t_i)$ List of words (lemmatized,without stop words nor punctuation) contained in the i-th HPO term. 31

$WE_{G1}(t_i)$ Embedding of the HPO term t_i obtained with Gen WE (version 1). 31

$WE_{G2}(t_i)$ Embedding of the HPO term t_i obtained with Gen WE (version 2). 31

$WE_{G3}(t_i)$ Embedding of the HPO term t_i obtained with Gen WE (version 3). 31

$WE_{LSA}(t_i)$ Embedding of the HPO term t_i obtained with DS WE (LSA). 30

$WE_{SVD}(t_i)$ Embedding of the HPO term t_i obtained with combined WE (SVD implementation). 32

$WE_{rand}(t_i)$ Embedding of the HPO term t_i obtained with a random WE. 40

$\hat{t}$ Final output prediction of a translation model, with $\hat{t} \in HPO$.. 34

t_{input} Input text of a translation model, either a layman term or a short sentence.. 34

$v(w_i)$ vector representation of the word w_i McDonalds' WE. 31

BioNLP Abbreviation for *Biomedical Natural Languages Processing*. 11

C Corpus of documents, each one representing an HPO term. The specific document of a term t_i is refered as $C[t_i]$. 30

C-LSTM Abbreviation for *Convolutional Long Short-Term Memory*. 27

C-LSTM-D Third MT model, with a convolutional net followed by a LSTM and a dense one. 35

C-NN Abbreviation for *Convolutional Neural Network*. 24

CBOW Abbreviation for *Continuous Bag Of Word*. 20

DA Abbreviation for *Domain Adapted WE*. 21

DAG Abbreviation for *Direct Acyclic Graph*. 15

DS Abbreviation for *Domain Specific WE*. 29

Gen. Abbreviation for *Generic WE*. 29

GRUs Abbreviation for *Gated Recurrent Units*. 25

HPO Abbreviation for *Human Phenotype Ontology*. 11

LSA Abbreviation for *Latent Semantic Analysis*. 19

LSTM Abbreviation for *Long Short-Term Memory*. 25

LSTM-D Second MT model, with one LSTM layer followed by a dense layer. 35

LSTM-P Fourth MT model, with a net trained for branch prediction that works in parallel with LSTM-D models. 36

MT Abbreviation for *Machine Translation*. 22

NLP Abbreviation for *Natural Languages Processing*. 11

NN Abbreviation for *Neural Network*. 20

R-NN Abbreviation for *Recurrent Neural Network*. 24

S4R Abbreviation for *Share4Rare*. 13

SVD Abbreviation for *Singular Value Decomposition*. 20

tf-idf Abbreviation for *Term Frequency-Inverse Documents Frequency*. 19

WE Abbreviation for *Word Embedding*. 19

1 Introduction

Nowadays, more and more people interact with computers: not only technicians and computer scientists, but also people with less technological literacy. For this reason, the way we communicate with computers is rapidly changing, adapting to users' necessities and becoming more accessible to everyone day after day.

In recent years, important breakthroughs have been achieved in order to make computers capable of understanding natural languages as we do. Indeed, many efforts are continuously made in order to break down the linguistic wall that separates humans and machines. The branch of Computer Science and Artificial Intelligence that studies how humans interact with computers using natural language is called **Natural Language Processing**, often shortened as NLP.

In general, NLP is not an easy task. Languages are more than a simple concatenation of words: understanding a sentence means understanding the single words that compose the sentence but it also means understanding how the concepts that those words represent are interconnected to create the idea behind the sentence. It is a process that includes several obstacles for a computer: words with ambiguous interpretations, methapors and figures of speech, irony, etc. are all aspects that humans can easily manage, but that can be insurmountable barriers for a computer.

The task of analyzing and understanding human language becomes even harder when NLP tecnhiques are applied in very specific fields, that often have their own vocabulary and where words have nuances that can be completely different form their general meanings. One of the most important examples of a specific area with its own language is the biomedical field. There exists a sub-field of NLP specifically dedicated to medical and biological languages, called **Biomedical Natural Languages Processing** (BioNLP).

One of the most studied applications of NLP (and BioNLP) is the translation between languages. In this case the effort of the computer is twofold: on one side, it has to understand what it has been said and, on the other side, it has to translate it into a correct sentence in a different language. Above the classical idea of translation, i.e. from a language to another one, when one is considering a specific field, the translation could be done from the specific vocabulary used in that field to the generic vocabulary, also called layman vocabulary. Especially in the case of the medical field, the differences between clinicians' and laymen's vocabularies are huge: most of the people that are not used to medical terminologies could find very complex to understand a medical test or, most importantly, they could not be able to describe their conditions appropriately.

The aim of this project is trying to fill the gap that exists between layman and specific terminologies. Several methods will be proposed to translate between layman terms and short sentences describing specific terms of the Human Phenotype Ontology (HPO).

1.1 Motivation

The gap between layman and technical language in the medical field is indisputable: most of the people describes their conditions using a general vocabulary and not the specific one, as can be easily ascertained checking the frequency of search for layman terms in one of the most common web search engines (*Google Search*) with respect to the corresponding medical term (Figure 1). This is an expected result: medical terms are often unknown or unusual for most of the people and it can not be expected for everyone to know the correct medical term to describe his symptoms. However, the complexity of the medical vocabulary is justified from the necessity of avoiding ambiguity and correctly identifying a symptom or a condition among plenty of similar ones. On the contrary, layman expressions are often ambiguous or even inaccurate and can bring to confusion and misunderstanding.

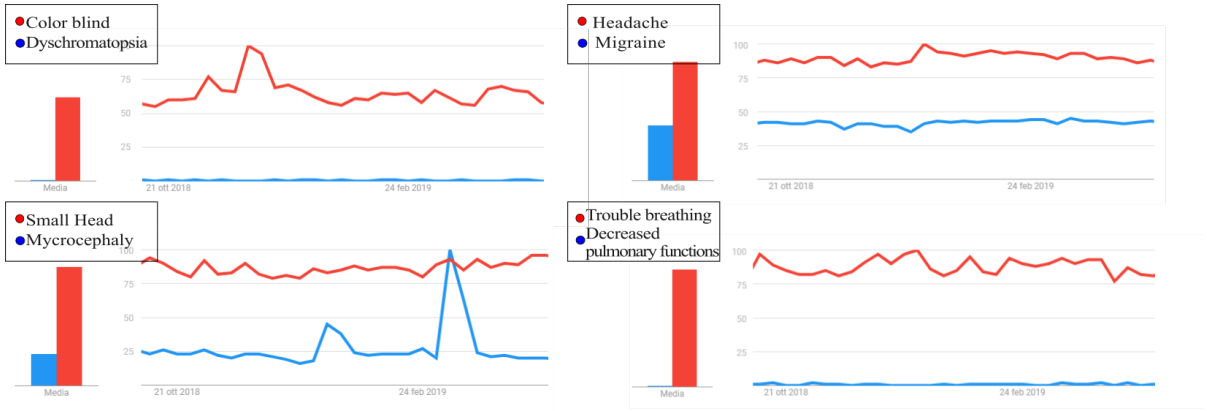


Figure 1: **Google searches in the last 6 months of layman terms commonly used to describe medical conditions (red) whit respect to the corresponding HPO concepts (blue)**(Images from: google.com/trends)

This difference between vocabularies becomes relevant when clinicians try to reconstruct the clinical phenotype of patients, i.e. the set of behaviors, physiological aspects and morphological characteristics that differentiate diseased and healthy subjects. The precise analysis of the clinical phenotypes of an individual is known as **deep phenotyping** and it includes medical history, physical examination, blood and laboratory tests and behavioral studies, among others. Data obtained through deep phenotyping has great potential to accelerate the identification of disease with prognostic or therapeutic implications[1]

Using precise phenotyping for diagnosis is of special relevance for the case of **rare diseases**, life-threatening diseases or chronically debilitating conditions that affect a limited number of people with respect to the entire population (less than 5 people per 10,000 inhabitants according to European legislation)[2]. Due to the limited knowledge about rare diseases, diagnosis is one of the most challenging tasks for a physician, often implying delays in the treatment or even wrong diagnosis and therapies. In an European survey involving 17 countries concerning eight of the most *common* rare diseases in EU, it was discovered that 25% of patients had to wait between 5 and 30 years from the appearance of the first symptoms to a correct diagnosis of their disease. Furthermore, in the first case the diagnosis received was wrong for 40% of the patients [3].

Share4Rare (S4R, www.share4rare.org) was born with the idea of helping patients affected by rare diseases: it is a collective online platform that aims to create a community where patients suffering from rare diseases, their caregivers, clinicians, and researchers will cooperate to generate new knowledge about these conditions. Patient users of the platform will provide clinical information about their condition i.e. diagnosis, symptoms, age of onset, etc. In return they will enter into a social media platform in which they will be able to access curated quality information, participate in different research initiatives, and track the results of their participation through charts and descriptive statistics. In a nutshell, patients will get access to information about their diseases that will improve their diagnosis and care process, while clinicians and researchers will collect data about patients' phenotypes, crucial for research and future diagnosis.

Inside a big cooperative project like this, a standardization of the medical terms that describe phenotypes becomes of central importance, and for this reason the Human Phenotype Ontology will be used. On the other hand, it can not be expected that patient users know this specific terminology. There it lies the necessity of a system that automatically translates users' layman terms used to describe their condition into HPO terms that can be appropriately stored in S4R databases.

1.2 Objectives and planning

The main objective of this project is to provide an instrument based on machine learning algorithms for the automatic translation between the layperson terminology with which common people describe their symptoms and a specific vocabulary of phenotypic abnormalities such as the Human Phenotype Ontology.

In order to achieve this main goal, some intermediate steps through the process of achieving the general objective should be defined, in particular:

- The design of specific text pre-processing algorithms, specifically designed for medical and HPO terms.
- The embedding of HPO terms into a vector space that maintains the semantic relationships existing between terms.
- The encoding of layman terms (after pre-processing) into a vector space.
- The creation of a function to map layman terms into the HPO vector space.
- The definition of metrics to evaluate results that go beyond a simple *correct/wrong* classification.

These intermediate goals, as well as the main objective of this project, were achieved during a period of 4.5 months through a well defined planning that included: a review of the state of the art and already existing solutions, the implementation of the proposed methods for both embedding and mapping, the evaluation of the models with cross-evaluation, and finally writing this document.

1.3 Outline

The rest of this report is dedicated to explaining how objectives listed in Section 1.2 were achieved, in particular Chapter 2 is dedicated to a small review of what an ontology is, with a particular focus on HPO: examples will be shown, trying to make the reader as familiar as possible with the architecture of this representation of human phenotypes.

After this introduction about HPO, in Chapter 3 the State of Art of the most relevant NLP techniques for automatic translation is described. Particularly, this chapter is divided in three sections that, respectively, summarize the most common techniques for representing words and sentences inside more complex machine learning algorithms, describe the machine learning techniques generally used in NLP, and introduce the techniques that can be used for the task of labeling synonyms with the corresponding HPO term.

Chapter 4 explained the methods that have been proposed to solve the problem of layman terms' translation. It is divided in two sections, one explaining the embeddings created to represent HPO, and the other explaining the machine learning strategies adopted to map layman terms in these spaces.

In Section 5, the experimental setup created to evaluate the methods of the precedent chapter are described. Particular attention is given to the creation of a training and test set as well as to the definition of the parameters that can be tuned and that can influence final results.

Chapter 7 is dedicated to a brief analysis of costs and resources needed to carry out this project, as well as a concise reasoning on the environmental impact that the project could produce.

Finally, Chapter 6 outlines the main results, trying to analyze the embeddings and the models separately and to evaluate the outcomes (i.e. the predictions over input layman terms) not only in terms of correct or wrong prediction, but with a more detailed analysis that tried to hold in consideration the semantic similarity between correct and predicted terms.

2 The Human Phenotype Ontology

An **ontology** can be defined as a formal and explicit description of concepts (also called classes) and the relationships between these concepts. For this reason an ontology is very useful to define a common and standardized glossary for researchers to share information in specific domains, limiting ambiguity [4]. Ontologies combine machine-interpretable definitions of basic concepts in a specific domain with relations between these concepts. The domain of the classes can vary a lot, from very general databases as WordNet [5] that aim to collect all English nouns, verbs, adjectives and adverbs, to more specific ones such as SNOMED CT [6], that contains clinical nomenclature, or the Gene Ontology [7], that aggregates genes and gene products.

A **phenotype** can be defined as the set of all the observable characteristics of an organism (or a cell). This includes its individual form, its functions and other characteristic aspects such as height, color, blood type and enzyme activity [8]. Therefore, the **Human Phenotype Ontology** (HPO) is an ontology dedicated to human phenotypes: it contains an analytic, complete and well-defined collection of more than 14,000 classes that describe human phenotypic abnormalities and more than 15,500 subclass relations between these classes. [9]

Each term in HPO describes a human abnormality through its technical names and other descriptors. In particular, terms are identified by a unique ID and a Label (i.e. its name). Most of the terms also contain a brief description of the term itself provided by clinicians or external databases and a list of synonyms obtained with various methods (see Section 3.3 for further details). Finally, HPO terms are linked to classes in other ontologies, for example to the corresponding class in SNOMED CT, or to the causal genes in the Gene Ontology. The relationship between terms is transitive and of type ***is-a***. As an example, consider the HPO term that refers to a particular malformation of the fingers, the *arachnodactyly*. The corresponding HPO class is:

> **Id:** HP:0001166
>
> **Name:** "Arachnodactyly"
>
> **Description:** "Abnormally long and slender fingers"
>
> **Synonyms:** "Long slender fingers"; "Long, slender fingers"; "Spider fingers"
>
> **Is a:** "Slender finger"; "Long fingers"
>
> **Xref:** MSH:D054119; SNOMEDCT_US:62250003; UMLS:C0003706 (*references to other ontologies*)

The transitive relationship between terms allow to structure HPO as a direct acyclic graph (DAG), improving the flexibility and descriptiveness of the ontology with respect to a simpler hierarchical graph (i.e. a tree). For example, the HPO term used as an example few lines above is both a *Slender finger* and a *Long fingers* abnormality: this means that this node has two parents in the ontology, that in turn could have more parents and so on, till reaching the unique root node.

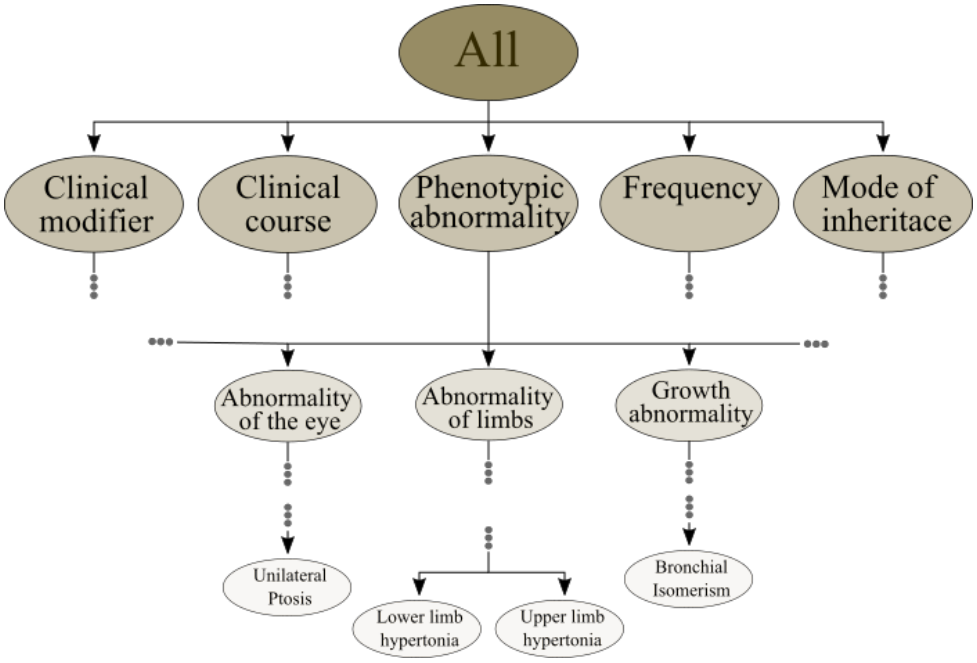


Figure 2: **HPO structure: from upper nodes to specific ones.**

In particular, as reported in Figure 2, HPO is divided in 5 sub-ontologies defined by the children of the root node of the ontology ("HP000001: *All*") at depth 1:

- **Phenotypic abnormality**: it is the main sub-ontology; it contains the description of the phenotypes collected in HPO.
- **Mode of inheritance**: it contains terms that describe the way a particular trait is passed through generations, e.g. *Mitochondrial inheritance* or *Autosomal dominant inheritance*.
- **Clinical modifier**: it contains terms that can be used to describe and characterize phenotypes in terms of severity (as *Severe*, *Moderate*, etc.), triggering factor (as *Triggered by cold*) and so forth.
- **Clinical course**: it describes how a particular abnormality could evolve, e.g. *Neonatal onset*, *Slow progression*, etc.
- **Frequency**: it represents frequency of specific phenotypes (*Occasional*, *Frequent*, etc.).

The structure of HPO allows us to define 29 categories (or **branches**) as the 25 descendants of the *Phenotypic abnormality* node plus the other 4 describing sub-ontology root nodes (*Mode of inheritance*, *Clinical modifier*, *Clinical course*, *Frequency*). The children of the *Phenotypic abnormality* class (that, as mentioned before, define most of the branches) at the first level refer to abnormalities of specific systems or body parts in general and they will contain most of the other HPO classes. Examples of these branches are *Abnormality of limbs*, *Abnormality of the cardiovascular system* or *Growth abnormality*. Since HPO is a DAG, each concept belongs at least to one branch (except for the root of HPO), with several terms belonging to two or more branches. The only exception is for a few terms labeled as *Obsolete* that stand for their own, without connections to other terms. Since obsolete terms do not contain anything but an ID and a label and, as the name suggests, they refer to terms that are no longer used, they will not be considered for the development of this work. Note that, from now on, when talking about HPO, it will be implicitly

considered a version of HPO without these terms.

The structure of the ontology can be used as well to define the **depth** of a term as the total number of hypernyms that the term has. Formally, given an HPO term t_i and the set of terms that are included in the path between t_i and the root node ($S(t_i)$), its depth is:

$$depth(t_i) = |S(t_i)| \tag{1}$$

Hence, for example, the depth of the root is $depth(\text{“}All\text{”}) = 0$ since it is the root node, the depth of a branch root is e.g. $depth(\text{“}Abnormality\ of\ the\ \ breast\text{”}) = 2$ and so on. The depth of the precedent example term is, for example, $depth(\text{“}Arachodactyly\text{”}) = 15$, while the maximum depth of a term is 42 (for the term *Synostosis of the proximal phalanx of the thumb with the 1st metacarpal*).

Finally, a function of **similarity** between two terms can be defined using the position of the terms in the ontology, as proposed by Seco et al. [10]. In particular, among all the alternatives, we decided to use Jiang and Conrath's formula [11] since Seco et al. demonstrated that it is the one that was more correlated with a similarity score proposed by experts. Thus, given two terms $t_1, t_2 \in HPO$, the similarity $sim(t_1, t_2)$ is defined as:

$$sim(t_1, t_2) = 1 - \frac{ic(t_1) + ic(t_2) - 2 \times sim_{res}(t_1, t_2)}{2} \tag{2}$$

Where $ic(t_i)$ is the information content of a term and $sim_{res}(t_1, t_2)$ is the Resnik similarity function [12]. Let $|HPO|$ be the total number of terms in HPO and $S(t_1, t_2)$ the set of terms that subsume t_1 and t_2, then $ic(t_i)$ and $sim_{res}(t_1, t_2)$ are calculated as:

$$ic(t_i) = 1 - \frac{log(depth(t_i) + 1)}{log(|HPO|)} \tag{3}$$

$$sim_{res}(t_1, t_2) = \max_{\tau \in S(t_1, t_2)} ic(\tau) \tag{4}$$

In what follows, similarity, depth, and branches of a term will be crucial in order to evaluate results.

3 State of the art

Natural Language Processing is the research field that studies how machines interact with natural languages, with the particular aim of understanding, analyzing and manipulating texts or speech [13]. It includes several areas of research, with methods and techniques from very different fields: statistics, computer science, information engineering, artificial intelligence, etc. The aim of this chapter is to briefly summarize the most important techniques introduced in NLP in recent years, with a particular focus on those techniques that could be more useful for the specific application of this work. A summary of the recent trend in NLP can be found in [14]. Moreover, an introduction to machine learning can be found in [15].

The rest of the chapter is structured as follows: at first a brief summary of the most used technique for word representation is given (Section 3.1); then Section 3.2 reports the machine learning schemes most widely used in NLP; finally, a description of the methods that can be used for labeling HPO synonyms is reported in Section 3.3.

3.1 Word Embeddings

Word Embedding (WE) refers to a series of different techniques where words from a vocabulary or short sentences are mapped in a vector space of N dimensions, with N usually varying between some hundreds to one thousand. Thus, a WE can be defined as an encoding of words in a relatively high-dimensional vector space [16]. It has been extensively shown that many WE are able to capture semantic properties and relationships between words and, as a consequence, WEs are the most common features used as input (or output) to machine learning models for many NLP tasks (e.g. information extraction, information retrieval, sentiment analysis, translating, question answering...) [17].

At first, the vector representation of words was based on statistical and frequency based techniques. The most basic way to do that is through **one-hot encoded** vectors: the vector size N corresponds to the size of the vocabulary and each dimension corresponds to one word. Vectors in the WE will contain a "1" in the position corresponding to the word that the vector represents. Limitations of this approach are evident: dimensionality of the embedding rapidly increases with the size of the dictionary, and to compare vectors is meaningless (except for equality checking).

A more useful technique is the **latent semantic analysis** (LSA) [18]: it takes a corpus of documents as input and it returns a vector representation of fixed dimensions of each word contained in the corpus. The first step is to build a term-document matrix which represents the frequency of words in the documents: one dimension of the matrix corresponds to documents and the other one to words; elements in the matrix are computed using some frequency statistic, such as the **term frequency-inverse document frequency** (Tf-idf) [19] index, calculated as the product of two different statistics. Given a word w, a corpus D and a document $d_i \in D$

$$tf\text{-}idf(w_j, d_i, D) = tf(w_j, d_i) \times idf(w_j, D) \tag{5}$$

Where:

- $tf(w_j, d_i)$ is the *term frequency* index, usually calculated as the number of times the word w_j appears in the document d_i, i.e. its frequency in the document i ($f_{j,i}$), normalized by the maximum frequency of a word in the document:

$$tf(w_j, d_i) = \frac{f_{j,i}}{\max_k f_{k,i}} \tag{6}$$

- $idf(w_j, D)$ is the *inverse document frequency* index: let n_j be the number of documents that contain the word w_j, over a total of N documents in the corpus, then:

$$idf(w_j, D) = log_2(\frac{N}{n_j}) \tag{7}$$

The tf-idf index of a word should indicate how much a word is characteristic of a document. Thus, a word that appears in almost all the documents will have a very low idf and, as a consequence, a very low $tf \cdot idf$. On the contrary a word that appears several times in only one document will have both high idf and tf and therefore, a high tf-idf.

The results of this first step of LSA is a matrix with as many rows as the number of words in the corpus, and as many columns as the number of documents:

$$X = \begin{bmatrix} tfidf(w_1, d_1, D) & tfidf(w_1, d_2, D) & ... \\ tfidf(w_2, d_1, D) & tfidf(w_2, d_2, D) & ... \\ ... & ... & ... \end{bmatrix}$$

Finally, the dimensionality of X is reduced via singular value decomposition (SVD), obtaining a new matrix X_k that approximates the information contained in X in a lower dimensional space, projecting the vectors associated with each word in a space of fixed dimension [20].

In recent years, the growing attention to machine learning techniques, alongside the increasing computational power of modern computers that allows the analysis of huge text corpora have put aside classical embedding techniques such as LSA, in favour of neural-network (NN) models. Among the huge number of techniques proposed in the last years (e.g Glove [21] or Senna [22]), the most popular is, without a doubt, **Word2Vec**[23], a vector representation based on two different NN models: the continuous bags of words (CBOW) and skip-gram models. Both models are based on the distributional hypothesis, i.e. the idea that words with similar meanings should appear in similar contexts in a text [24]: based on this idea, CBOW calculates the probability of a target word conditioned on a given set of context words that surround the target word across a window of size k. On the other hand, the skip-gram model behaves the opposite way: it predicts the probability of the surrounding context words given the central target word [14].

As reported in Figure 3, CBOW consists of a simple fully connected NN with one hidden layer. The inputs of the model are usually one-hot vectors representing context words, thus it has V neurons per each context word, where V is the vocabulary size; the hidden layer has N neurons and the output layer, that represents the *softmax* probability over all the words, has again V neurons. The weight matrices of the hidden and output layers, $W \in \mathbb{R}^{V \times N}$ and $W' \in \mathbb{R}^{N \times V}$,

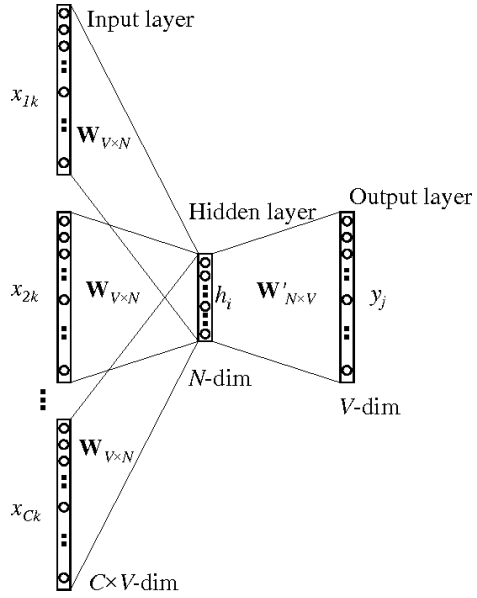


Figure 3: **CBOW model** (image from [25])

are used to extract the vector representations of the words: the i^{th} word of the vocabulary is represented by:

$$v_c = W_{(i,:)} \qquad v_w = W'_{:,i}$$

The Skip-gram model is very similar, but with the context words as output and a single word as input.

In addition to the fact that it maps words that appear in similar context close, Word2Vec encodings combined through linear operations result in vectors that are semantic composites of its components [14][26], for example:

$$vec(\textbf{king}) - vec(\textbf{man}) + vec(\textbf{woman}) \approx vec(\textbf{queen})$$

$$vec(\textbf{Germany}) + vec(\textbf{Capital}) \approx vec(\textbf{Berlin})$$

A big limitation of word embeddings created with neural models is that they must be pre-trained with large corpora that influence the final representations: as a consequence, a domain specific vocabulary with a small training corpus will probably not be well represented in the vector space. On the contrary, techniques such as LSA are created based on the specific corpus and could better capture domain specific semantics, even if the vector representation is not as accurate as the one obtained with Word2Vec or similar techniques. In literature, there are several works that try to concatenate different embeddings in order to capture the different strenghts of the models. W. Yin and H. Schütze [27], for example, tested several approaches: from a simple concatenation of the vectors to a NN model that mixes the different embeddings. With the same aim, Sarma et al. [28] created a **Domain Adapted** WE (DA): they used non-linear kernel canonical correlation analysis to create a new embedding from a generic embedding such as Word2Vec and an embedding created with LSA.

The problem of correctly representing the semantic content of words in a specific domain was also addressed by M. Pilehvar and N. Collier [29] with an embedding specific for HPO terms. They first use BabelNet [30], a big semantic network that merges knowledge from WordNet and Wikipedia, to extract concepts from each HPO term (e.g. from the HPO term t_i =*flexion contracture of digit* to the list of concepts $C = \{flexion,\ contracture,\ digit\}$). They then extract the Wikipedia pages of each concept, and from them they get the ordered list of the most specific words in comparison with all the Wikipedia articles, $R_i = [w_1, w_2, ..., w_n]$. Finally, they used a pre-trained WE such as Word2Vec to create the representation of the HPO terms. Given $V(w_j)$ the vector representation of the j^{th} word in the order list R_i, the vector representation of HPO_i is :

$$V(HPO_i) = \sum_{w_j \in R_i} e^{-\lambda j} V(w_j) \tag{8}$$

With λ being a decay factor fixed to 0.2.

3.2 Deep learning in NLP

The problem of mapping layman terms to the corresponding HPO terms can be seen, in a nutshell, as a translation problem between a specific language (the HPO vocabulary) and a layperson one. Machine translation (MT) has been studied since the 1950s [31], and still remains a topic of interest in the research community, as demonstrated by the increasing trend of publications on the matter (Figure 4).

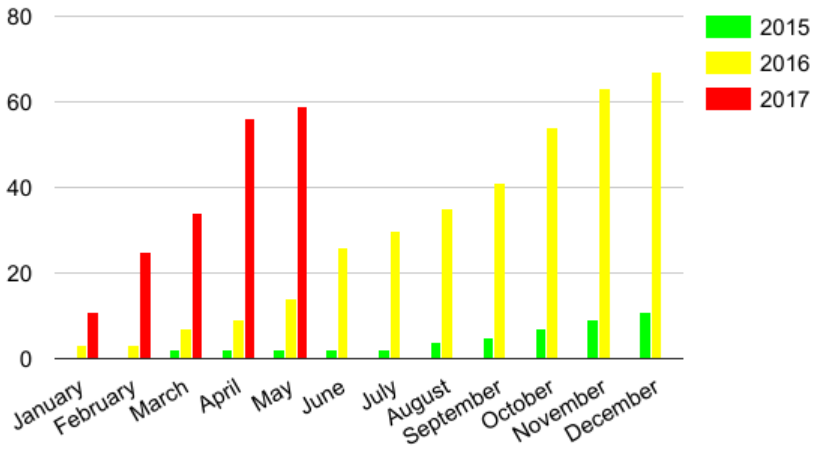
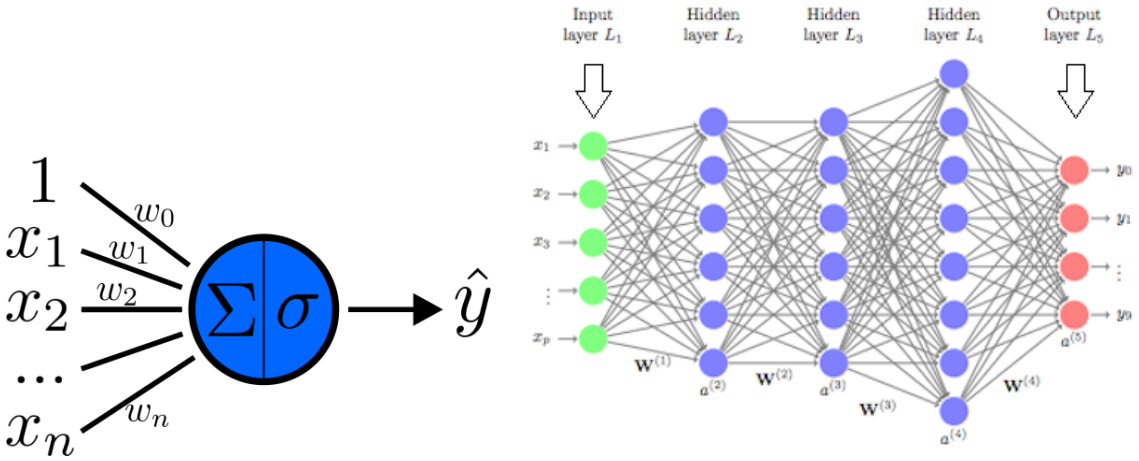


Figure 4: **MT related papers published on arXiv** (Image from *proz.com*)

Another way to address the issue of layman translation is to consider it as a text classification problem, where each entry (layman terms) should be classified into a specific category (HPO terms).

Both MT and text classification tasks have experimented a step forward in the last years, due to the increasing interest of the research community in these topics and to the introduction of **deep learning** architectures. Deep learning is a diversified group of machine learning techniques

that share one characteristic: the use of several layers of non-linear functions trained on huge datasets. They are used for a wide range of applications: supervised or unsupervised feature extraction and transformation, pattern analysis, classification, etc. [32].



(a) Artificial neuron

(b) Fully-connected deep neural network example (image from orbograph.com)

Figure 5: **Deep neural network basic functioning.**

The computing unit of deep learning models is an artificial neuron, depicted if Figure 5.a. The input of a neuron is a vector $\mathbf{x} = [x_1, x_2, ...x_n]$ and the output, y is a combination of the values in x:

$$y = \sigma(\sum_{j=1}^{n} w_j x_j + b) \tag{9}$$

Where $\sigma(\cdot)$ is the *activation function* (common activation function are *linear*, *relu*, *sigmoid*, etc.), and $w_j\, j = 0...n$ are the weights, that in the networks play the role of *trainable parameters*. Finally, b is the bias term that is also a trainable parameters. Neurons are trained using *loss functions* that define an error between the predicted output ($\widehat{y}$) and the correct output y. A common loss function is $l = \frac{1}{2}(\widehat{y} - y)^2$, but there are plenty of alternatives. Weights are iteratively adjusted during the training phase trying to minimize this loss function.

Neurons are connected in several layers to create a deep neural network, as reported in Figure 5.b. The basics functioning of each neuron is the same as before, hence the output of the $i-th$ layer ($\mathbf{a}^i$) can be calculated as:

$$\mathbf{a}^i = \Sigma^i(W^i \cdot \mathbf{a}^{i-1} + b^i) \tag{10}$$

$$W = \begin{bmatrix} w_{11}^i & w_{12}^i & ... & w_{1n}^i \\ ... & & & \\ w_{m1}^i & w_{m2}^i & ... & w_{mn}^i \end{bmatrix} \tag{11}$$

Where w_{ij}^l is the weight that connect the j-th neuron in the layer $l-1$ to the i-th neuron in the layer l and Σ^i idicates the activation functions of layer i.

This simple scheme can be modified in several ways, creating different models. Among the vast amount of possible architectures proposed in the last years, there are two solutions that have become widespread for deep learning in NLP : **convolutional neural networks** (C-NN) and **recurrent neural networks** (R-NN).

3.2.1 C-NN for NLP

The weight matrices in the architecture seen before have a number of parameters that tend to explode when the dimension of the input increases. Convolutional neural networks overcome this issue introducing some filters that reduce the number of weights sharing them between the input elements of a layer.

In the discrete domain, the convolution between two functions f and h is define as:

$$(f * h)(m) = \sum_{n=-\inf}^{\inf} f(n)g(m-n) \tag{12}$$

For example, in the case of a 2D input (as an image), h is a filter of dimensions $[2a * 2b]$, and Equation 12 becomes:

$$(f * h)(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} h(s, t)f(x-s, y-t) \tag{13}$$

Usually C-NN are composed by several banks of filters composed by trainable weights that extract features from the input, followed by layers that select only the most salient features (e.g. *max-pooling* layers).

Given their effectiveness in computer vision tasks, C-NNs were also applied to solve NLP probems, with optimal results principally in texts and sentences classification tasks ([33], [34]). The basic model for the application of C-NN in NLP was deeply analyzed by Y. Zhang and B. Wallace [35] as reported in Figure 6.

The main idea of this approach is to embed each sentence in a matrix of fixed dimensions the rows of which are vector representations of the words of the sentence itself, obtained with a WE technique. It is now possible to effectively deal with this *sentence matrix* as usually C-NNs deal with images, that is, it is possible to perform convolution on it via linear filters [35]. A common strategy is to apply a bank of filters of different sizes that can extract features from the sentence matrix. Each feature then passes through a *max-pooling* function to reduce the dimensionality and create a unique feature vector of fixed dimensions. Finally, this feature vector is used to feed a *softmax* layer that maps it in a specific category.

This simple architecture, with some modifications, has been demonstrated to achieve good results in several NLP tasks, from Part-of-Speech taking [36] to sentiment classification, from semantic related words identification to MT tasks [37]. In particular C-NNs, thanks to the convolutional filters, perform very well in extracting features at different positions of the sentence,

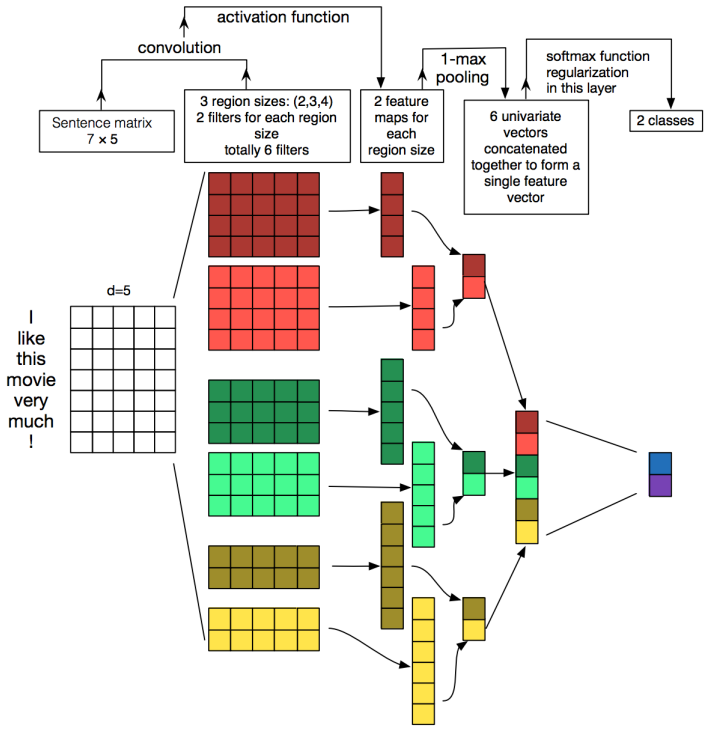


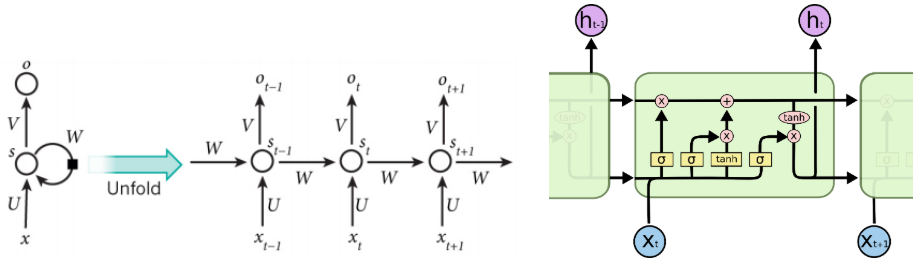
Figure 6: **C-NN architecture** (Image from [35])

even if they are not able to capture correlation between words in different positions in the input sentence. To resolve this limitation, R-NNs have been introduced in NLP.

3.2.2 R-NNs for NLP

As mentioned before, a big limitation of C-NNs is their inability to model long distance dependencies between words as well as their position inside the sentence. Furthermore, C-NNs are unable to model texts of different lengths. For these reasons, R-NNs, that are able to process sequential information, are better suited for their use in many NLP tasks, first and foremost for MT.

The basic architecture of a R-NN is reported in Figure 7.a: the input of the net at time t, x_t is a vector representation (a WE) of the t^{th} word of the input sentence. At each time step there is a hidden state $s_t = f(Ux_t + Vs_{t-1})$ that depends on the current input and on the precedent state. f is an activation function and U, V, W are weight matrices shared across the different time steps. The output of the net is o_t and it is often obtained applying a non-linear transformation. Since the hidden state is propagated to future steps, it is considered to be a kind of memory element, that accumulates knowledge about the previous inputs. Classical R-NNs are difficult to train, due to the *vanishing gradient* and the *exploding gradient* problems. In NLP, these issues have been overcome with two R-NN variants : **long short-term memory** (LSTM) [39] and gated recurrent units (GRUs)[40].



(a) *R-NN architecture* (Image from [38]) (b) *LSTM block* (Image from *colah.io*)

Figure 7: **R-NN architecture and LSTM block**

LSTM, the structure of which is reported in figure 7.b, introduces 3 gates that are combined to calculate the output: the *input gate* (i_t) decides how much an input at time step t influences the hidden state; the *forget gate* (f_t) regulates the update of the precedent hidden state in the current one, being able to leave aside some values; finally, the *output gate* (o_t) decides how much of the current state is passed to the next time step [41]. The equations that regulate these three gates and the hidden state/output calculations are:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{14}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot tanh(W_c[h_{t-1}, x_t] + b_c) \tag{15}$$

$$h_t = o_t \cdot c_t \tag{16}$$

GRUs blocks are very similar to LSTM, but with only two gates (named *update gate* and *reset gate*), the function of which is comparable to LSTM gates' function. The first gate operates as the forget and input gate of an LSTM: it decides what new information to discard and what new information to keep. The second gate, on the contrary, decides what information from the precedent time step to preserve.

Many applications have been found for LSTMs and R-NNs in NLP; for instance the **encoder-decoder model** proposed by Sutskever et al. [42] has rapidly become the state of art in machine translation tasks as well as in different tasks that required text generation [43]. The main idea of the model, reported in Figure 8, is to use a chain of LSTM blocks to encode the input sentence in a vector of fixed dimensionality, using then a second LSTM chain to decode the target sequence from the vector. Authors also demonstrated that the vector produced by the encoder part can be considered as an embedding of the input sequence, with sentences with similar meaning forming clusters in a vector space of fixed dimensionality.

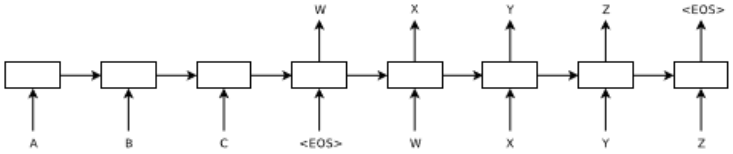


Figure 8: **encoder-decoder architecture** (Image from [42])

We have seen that C-NNs are able to learn local responses from temporal or spatial data but lack the ability of learning sequential correlations. On the other hand, R-NNs can model sequential correlations but are unable to extract features in a parallel way [44]. Aiming at integrating the two models in order to benefit from strengths of both, Zhou et al. [44] proposed a **Convolutional LSTM model** (C-LSTM). This architecture is composed of two main blocks: first, a bank of convolutional filters is applied to the sentence matrix. The aim of this first step is to extract n-gram features at different positions of the sentence; second, the features extracted with the filters are used to feed an LSTM recurrent net, in order to capture long term dependencies between features. C-LSTM appears to improve both LSTM and C-NN results.

3.3 HPO translation

As far as I know, there is no work explicitly focused on automatically mapping layman terms and in general small texts into specific medical concepts. Nevertheless, to try to spot similar sentences in texts is not a new topic, and various solutions were proposed among the years.

V. Vydiswaran et al.[45] proposed a **pattern-based method** to catalog couples of consumer health terms and professional expressions from Wikipedia, "a large text corpus created and maintained by the community". They identified a series of phrases that usually link terms with their synonyms, e.g *also called*, *commonly known as*, *sometimes referred to as*, etc., and they used these phrases to identify medical-variant term couples inside the Wikipedia Health and Medicine corpus. Their method was able to identify synonym pairs in texts and, using the frequency of the term inside Wikipedia to understand whether it was a common term or a medical term, automatically labeling them as either specific or layman with good results.

In general it has been widely demonstrated that co-occurrence counts of pairs of words and other contextual information extracted from large corpora can be used for spotting synonym couples: M. Baroni and S. Bisi [46] for example demonstrated that the **mutual information** between couples of words obtained using co-occurrence count in a very big corpus can recognize couples of semantically related terms with very high accuracy. Similarly, Hagiwara et al. [47] used words dependencies (as subject-object relationship), sentence co-occurrence, and proximity for automatic synonym acquisition from general corpora.

Soğancıoğlu et al. developed **BIOSSES**, a "web-based system for biomedical semantic sentence similarity computation" [48]. Specifically, in this work authors analyzed the most common approaches for sentence semantic similarity computation, showing that, in the biomedical domain, most of these approaches produce poor results and seem to not be able of encompassing the actual biomedical knowledge [48]. The approches they analyzed can be divided in three main areas:

- **String similarity measures**, i.e. methods based on characters and terms similarity functions, as **Q-gram similarity** [49] or **Overlap coefficient** [50].
- **Distributional vector model**, i.e. measures based on a vector representation of sentences (a sentence embedding).
- **Ontology-based similarity**, i.e. the usage of ontologies (WordNet [5] and UMLS[51]) to calculate words distances used to get sentences similarity measures.

Finally, the authors introduced a regression model that exploits the prediction of the precedent approaches. With this new method, satisfying results in sentence-level semantic similarity computation were achieved.

Results comparable with the ones of BIOSSESS were obtained by Q. Chen et al. with the embedding **BioSentVec**[52]: a "sentence embeddings trained with over 30 million documents from both scholarly articles in PubMed and clinical notes in the MIMIC-III Clinical Database" [52], created using the **sent2vec** model [53] . It is the first publicly available sentence embedding in the biomedical field. Authors tested BioSentVec using the same test set used to evaluate BIOSSESS, outperforming results that Soğancıoğlu et al. obtained with their general sentence embedding and reaching results similar to the ones of the regression model. Doing so, they demonstrated the importance of the training corpus to build vector representations of words: a sentence embedding trained on domain specific corpora can achieve state of the art results in sentence similarity tasks, while the same embedding trained on a general corpus achieves unsatisfactory results.

Also the HPO specific word embedding proposed by M. Pilehvar and N. Collier [29], obtained with linear combinations of embeddings of salient words related to each HPO term (see Section 3.1 for more details) can somehow capture the relationships between layman and HPO terms: in order to have a qualitative evaluation of the embedding, synonyms were mapped in the space created as the semantic representation of HPO, and it was found that when a domain specific WE was used for representing individual words, the correct HPO term associated to the synonym was the closest point in the vector space in more than 35% of the terms tested. Considering that finding the correct HPO class of a layman term corresponds to picking up a term from a list of more than 14,000 possibilities, this result shows that an appropriate embedding is crucial for the proposed task.

It is worth remembering that most HPO classes already contain synonyms and layman terms: thanks to the works of N. Vasilevsky, S. Köhler, P. Robinson et al. [54], [55], [56] at the moment HPO contains more than 17,500 synonyms, of which 45% are labeled as "layman term". Furthermore, 57% of the HPO terms contain at least one synonym, and 35% of them have at least a layperson term, classified as *Exact*, *Narrow*, *Broad* or *Related*. This remarkable work of labeling synonyms to HPO terms was done systematically using different methods, from checking online knowledge such as Wikipedia and MedlinePlus, to using different ontologies, terminologies, and texts (e.g. mapping SNOMED CT terms into HPO, see [57]).

4 Methods

As seen in Section 3.3, there is no standard solution to the synonym identification problem. Furthermore, there seems to be a lack of studies when the problem is restricted to a very specific field as it is the case of HPO terms. Methods based on statistics and counts of words can not work in this case, due to the limited corpora available for some HPO classes: since some terms are very specific, it could be very difficult to find texts that describe them and provide synonyms and layman words at the same time. The high specificity of the terms makes the usage of tools for similarity calculation such as BIOSSES or BioSentVec inappropriate as well, since many sentences are not represented.

As shown in Figure 9, the proposed method is based on two steps: On the first place, a vector space to represent HPO terms, an HPO embedding, was created; then, I trained an algorithm to map layman terms and other text descriptors in general into this space.

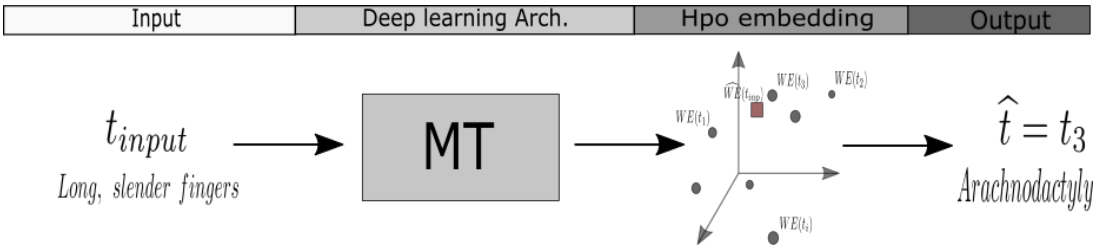


Figure 9: **General architecture for synonyms identification.**

The rest of the chapter is dedicated to the explanation of how the HPO embedding and the MT net have been implemented.

4.1 Word Embedding

As seen before, there are many possibilities for creating an embedding: following the nomenclature of Sarma et al. [28], it is possible to divide the embeddings in three categories:

- **Domain specific WE** (DS WE), is a category of WEs created from a specific corpus with, for example, statistical techniques.
- **Generic WE** (Gen. WE), embedding trained on huge corpora, such as Word2Vec, Glove, etc.
- **Combined WE**, created aligning generic and domain-specific embeddings, for instance Sarma DA WE (see Setion 3.1).

In Section 3.1, we have seen that DS WEs are usually unable of representing words as well as Generic embeddings but, when the application field is very specific, they could capture semantic nuances that other techniques can not. On the other hand, a combination of the two techniques could potentially overcome both DS WE and Gen. WE, even if combining multiple embeddings is not a straightforward task. Since a correct representation of HPO could be cru-

cial for a good translation, I studied the three techniques separately, trying to understand their drawbacks and usefulness for the task at hand.

4.1.1 Domain Specific WEs

Due to its ease of implementation and its widespread use before the introduction of the Gen. We, I decided to test the **LSA embedding** ($WE_{LSA}(t_i)$). A generic introduction to this technique was given in Chapter 3, however I should clarify how I implemented it for the specific application to HPO classes.

The first step is to define a corpus C of documents each one representing an HPO term: for each class I extracted name, description, synonyms and parents and I concatenated them in a unique body of text. Then, I cleaned each text eliminating punctuation and replacing numbers with their analogous in letters (for further details about the cleaning algorithm see Chapter 5 and Algorithm 1 in Appendix A). Finally, each cleaned text was used as a document to represent the corresponding HPO concept. An example of the creation of a document for an HPO class is reported in Figure 10.

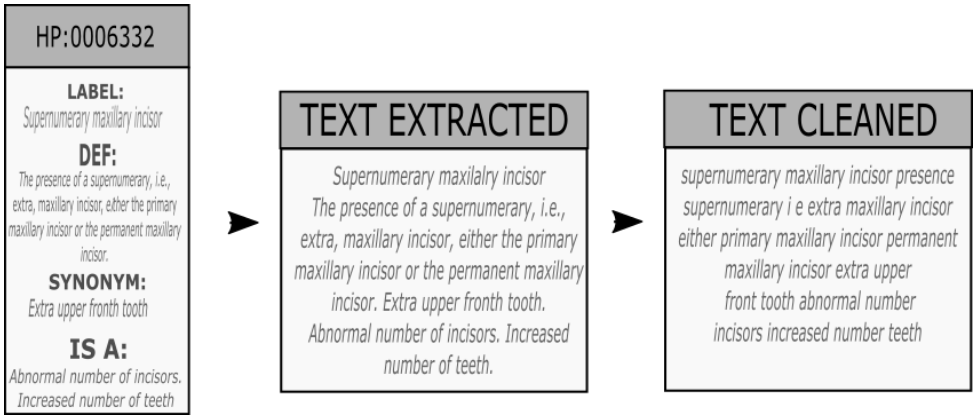


Figure 10: **Example of document extraction for LSA.**

The second step is to create a matrix X whose rows represent HPO concepts and columns represent words of the corpus. For example from the HPO concept $t_3 = HP:0000003$ ='*Multycistc Kidney Displasya*' I got the document $C[t_3] =$ {'*multicystic kidney dysplasia multicystic kidneys multicystic dysplastic kidney multicystic renal dysplasia renal cyst*'} and consequently the row of X corresponding to t_3 contains all zeros except for the 7 tf-idf indices of the seven unique words of $C[t_3]$:

$$X[3, multicystic] = 0.85 \quad X[3, kidney] = 0.28 \quad X[3, dysplasia] = 0.26 \quad \dots$$

Where $X[i, w]$ indicates the element of the matrix X in the row i^{th} and in the column corresponding to the word w.

Each line of X is already a representation of the corresponding HPO term, but in a space of more than 9,000 features (the cardinality of the corpus of document); after feature reduction via

SVD to obtain a fixed number of columns k, the rows of the new matrix are used as WE:

$$WE_{LSA}(t_i) = X_k[i,:] \tag{17}$$

With X_k the matrix obtained after SVD dimension reduction and $X_k[i,:]$ the i^{th} row of this matrix. Both $tf\text{-}idf$ indices and SVD were calculated using Scikit-learn Python library [58].

4.1.2 Generic WEs

Among the various possibilities for a Gen. WE, I decided to use an implementation of an embedding based on Word2Vec. Since the training of this model requires a big amount of resources, including time, RAM, data etc., I preferred to use a pre-trained model. McDonald et al. [59] published a **Word2Vec model for biomedical applications**: this WE was created using a skip-gram model with a window size of 5 and with 2 versions, one with an embedding dimension of 200 and the other with an embedding dimension of 400. They trained the model with more than 27M of biomedical articles from MEDLINE/PubMed.

HPO terms are, more than single words, concepts described by a sentence (e.g. *Atrial septal defect, Abnormal mitral valve morphology, etc*) thus, to obtain an HPO representation, vectors of the McDonald's WE should be somehow combined, even if in literature there does not exist a standard method to combine multiple embeddings [28]. I implemented three alternatives, denominated $WE_{G1}(t_i)$, $WE_{G2}(t_i)$ and $WE_{G3}(t_i)$.

The first version of the Gen. WE is motivated by the compositionality of Word2Vec, i.e. by the fact that the sum of different word vectors results in a vector that represents a word that is semantically related with the words represented by its component vectors. For each HPO term t_i, I extracted the list of words (lemmatized, without stop words nor punctuation) contained in t_i, $R_t(t_i)$= $\{w_1, w_2, .., w_n\}$ (for more details on the construction of R_t see Algorithm 2 in Appendix A). Then, the vector representation of t_i, is just the sum of the vector representation of the words w_i $i = 1, .., n$ in McDonald's WE, $v(w_i)$:

$$WE_{G1}(t_i) = \sum_{w_j \in R_t(t_i)} \mathbf{v}(w_j) \tag{18}$$

WE_{G1} considers each word in R_t the same way, no matter how much influence it has on the semantics of the HPO term. For example, for the HPO term *Abnormal mandible coronoid process morphology*, it might make sense to give a higher weight to the words *mandible* or *coronoid*, rather than to the words *abnormal* or *process*. To do so, an option is to exploit the X matrix created for the DS embedding (WE_{LSA}), since the $tf\text{-}idf$ index reflects the importance of words in documents. Thus, this second version of Gen. WE was obtained summing the vectors of the words in $R_t(t_i)$ multiplied by the tf–idf index of each word:

$$WE_{G2}(t_i) = \sum_{w_j \in R_t(t_i)} tf\text{-}idf(w_j, C[t_i], C) \cdot \mathbf{v}(w_j) \tag{19}$$

The last strategy is the most elaborated one and is based on Pilehvar's HPO representation introduced in Section 3.1, even if some modifications were introduced to avoid the usage of BabelNet (since it is not open source). Again, the starting point is the list of words contained in each term R_t. The first step is to build a corpus of Wikipedia pages related to each term ($D_t(t_i)$) according to the Algorithm 3 in Appendix A: for each word $w_j \in R_t(t_i)$, the first n pages of Wikipedia that are found searching for it are collected, among with the first l linked pages for each one of the n pages.

After this, D_t is used to extend $R_t(t_i)$, obtaining $R_t^*(t_i)$, the sorted list of words of $D_t(t_i)$ that have a lexical specificity higher than a given threshold (see Algorithm 4 in Appendix A). Lexical specificity [60] is a measure of the importance of a word in a corpus: given a word w, its frequency in a corpus (Wikipedia in this case) F, its frequency in the sub-corpus (D_t in this case) f, and the total number of words in corpus and sub-corpus respectively T and t, then the specificity of w is:

$$spec(w) = -log_{10}P(X \geq f) \tag{20}$$

with X being a hypergeometric distribution of parameters F, T, t. Thus, $R_t^*(t_i)$ contains in the first positions the words of $R_t(t_i)$ and, afterwards, lemmatized words of $D_t(t_i)$ that have a specificity higher than a given threshold ($th = 5$), filtered with a medical dictionary (the *Hunspell English medical dictionary*, 2017), and ordered according to their specificity. Eventually, the embedding is obtained summing the words in R_t*, with words that come from Wikipedia multiplied by a decay factor λ:

$$WE_{G3} = \sum_{w_j \in R_t^*(i)} f(w_j) \cdot \mathbf{v}(w_j) \tag{21}$$

with:

$$f(w_j) = \begin{cases} 1 & if \ \ w_j \in R_t(i) \\ e^{-\lambda \cdot j} & if \ \ w_j \notin R_t(i) \end{cases} \tag{22}$$

Even if the base WE used to calculate the vectors $\mathbf{v}(w_j)$ was trained on a medical corpus, not all the words needed were represented: specific HPO concepts as *hypoleucinemia* or *polyminimyoclonus* have no vectors in the Word2Vec model used. As far as possible, these words were replaced by synonyms or periphrasis that had the closest possible meaning. This was done only for the words in R_t, and not for all the words from Wikipedia in R_t^*. A list of the original words together with the proposed alternatives is reported in Tables 10 and 11 in Appendix B. Another problem relative to the creation of R_t was the presence of words separated by a hyphen, as *beta-cell* or *atlanto-occipital*. In this case, if the couple of words was present in the McDonald WE, then it was considered as a unique word in R_t. On the other hand, coupled words not represented in the WE at hand were split and inserted in R_t as two different words.

4.1.3 Combined embedding

In the work of W. Yin and Schütze [27], several techniques to combine different embeddings were proposed and tested. One of the most effective one was based on dimensionality reduction through SVD, and, also due to its ease of implementation, it is the one that was used in this work. Also note that the DA WE of Sarma et al. is a way to combine WEs, but it is complicated to implement, and thus it was not tested. Given an HPO term t_i, the combined WE ($WE_{SVD}(t_i)$) is obtained in two steps: first, a DS WE and a Gen. WE are concatenated to obtain a matrix

$M \in \mathbb{R}^{|HPO| \times (|WE_{Gi}|+|WE_{LSA}|)}$ in which each row represent an HPO term and each column a feature of one of the two embeddings:

$$M = \begin{bmatrix} WE_{Gi}(t_1) & WE_{LSA}(t_1) \\ WE_{Gi}(t_2) & WE_{LSA}(t_2) \\ \dots & \dots \end{bmatrix} \tag{23}$$

The second step is to reduce the dimensionality of this matrix via SVD, similarly to what done for the DS WE (WE_{LSA}), and to use the rows of this new matrix M_k as a vector representation of the HPO terms. Following the same nomenclature of equation 17, the embedding of a term t_i is:

$$WE_{SVD}(t_i) = M_k[i, :] \tag{24}$$

4.2 Machine translation

Once a vector space containing HPO terms has been created, the next step is to map inputs into this space. In this work 4 different deep learning architectures were investigated. For all these models, the encoder-decoder architecture of Sutskever et al. [42] was the backbone. The rest of the chapter is dedicated to explain how they work, trying to justify the process that led from the first model, that is almost equal to the classical encoder-decoder architecture, to the last more complex model.

4.2.1 Encoder-Decoder model

The main difference between a normal MT task and the task of spotting synonyms from a predefined list of terms is that in the first case the output could be any of the several combinations of words of the output vocabulary, while in the second case the possible combinations are limited to the ones that form an HPO term. The first architecture proposed to achieve this task is represented in Figure 11. As one can note, it is very similar to the encoder-decoder model reported in Figure 8: the main idea is to translate the input from the *layman language* into the *HPO language*, and then pick up the HPO term that is more similar to the predicted output.

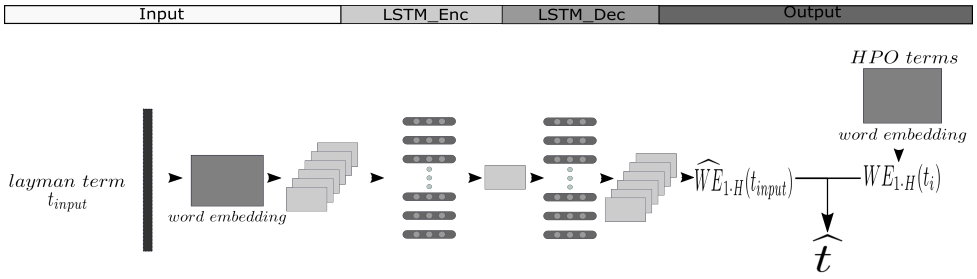


Figure 11: **Encoder decoder model for layman-HPO translation.**

The first step is to make the input text (t_{input}) *readable* for the successive layers: this is done using a **Tokenizer** class. Each input sentence was turned into a sequence of integers with each integer being the index of a token in a dictionary that contains all the words in the layman input corpora. The dictionary is ordered by frequency, i.e. very common words have a low index while specific words have a high index.

This vector can then be used to feed the actual network. The first layer is an **embedding** that transforms the input vector in an *embedding matrix*, each row representing a word in the original input. It is worth noting that this embedding layer is completely separated from the precedent HPO embedding: the later is a vector space that represent HPO concepts, while the former is a way to represent input words in the most optimal way for their use in the following layers. The HPO embedding is created once at the beginning of the process and thus it remains the same during all the process, including the training phase, while this embedding layer is trained with the model and its weights are continuously updated during the training phase. Each row of this embedding matrix output of the embedding layer is then used to feed a LSTM block of a RNN (**the encoder**) that analyzes the input sentence and maps it in a vector of fixed dimensionality. This vector finally feeds another LSTM, (**the decoder**), that tries to predict the HPO term.

Each block of the decoder LSTM has a vector as output, corresponding to a one-hot encoded embedding of a word in the HPO dictionary, thus the output of the model is another embedding matrix $M(t_{input}) \in \mathbb{R}^{l_{inp} \times |HPO|}$ with l_{inp} the length of the input sentence. In order to select an HPO term the rows of the output matrix were concatenated in a unique vector $\widehat{WE}_{1 \cdot H}$:

$$\widehat{WE}_{1 \cdot H}(t_{input}) = \left[M(t_{input})_{1,:}, M(t_{input})_{2,:}, ..., M(t_{input})_{l_{inp},:}\right] \tag{25}$$

Being $M(t)_{i,:}$ the i^{th} row of the matrix $M(t)$.

At the output $\widehat{WE}$ is compared with the vectors representing HPO terms obtained with the same procedure, i.e. one-hot encoding of the words to obtain $M(t_i)$ and concatenation of the vectors getting $WE_{1 \cdot H}(t_i)$ for each HPO term t_i. The closest HPO term (using cosine similarity or Euclidean distance) was then selected as final prediction $\hat{t}$:

$$\hat{t} = arg \min_{t_i \in HPO} \left(dist(\widehat{WE}_{1 \cdot H}(t_{input}), WE_{1 \cdot H}(t_i))\right) \tag{26}$$

Note that using this architecture the HPO WEs explained in Section 4.1 were not used.

4.2.2 Encoder-Dense Model

The precedent model is the classical architecture used for MT tasks, with an additional step that forces the output sentence to be an HPO term. Since the only vector representation used is the one-hot encoding, the model takes no advantage from information about semantic similarity of HPO terms. Another drawback of the encoder-decoder model is that there are two different LSTM layers with a big amount of both RAM and CPU/GPU usage for training.

For this reasons, the idea behind the second proposed model, further on named as LSTM-D, is to use a single LSTM layer that, as demonstrated in Sutskever's work, can create a vector representation of the input layer. Then this vector is directly mapped in the HPO space, without a LSTM layer that acts as decoder to create a sentence from it.

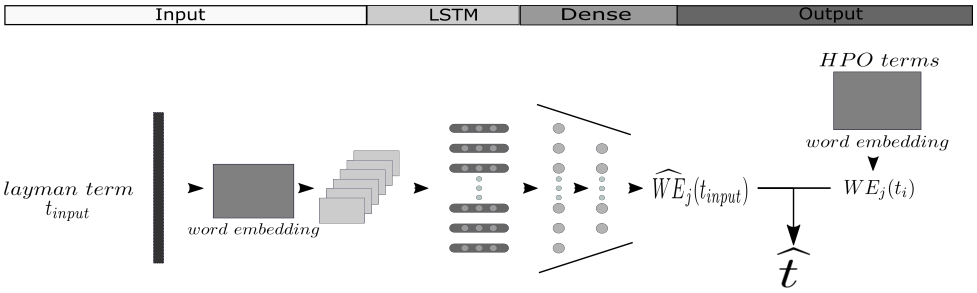


Figure 12: **LSTM-D model for layman-HPO translation.**

As reported in Figure 12, the first part of the model is equal to the one of the encoder-decoder model: a numerical representation of the input sentence feeds an **embedding layer** that, in turn, feeds a **LSTM**. Again, the output of the embedding layer is an embedding matrix $L \in \mathbb{R}^{l_{inp} \times N_{WE}}$ representing the input sentence, where N_{WE} is the length of the embedding, with each row representing a word in the input term t_{input}. Each one of these rows is the input of a different LSTM block in the R-NN layer. The output vector of the last LSTM block then enters in a classical **fully-connected layer** (dense layer) that simply maps it from the output space of the LSTM into the HPO space.

Let j be the HPO space (e.g. $j = G1\ A, LSA\ B, etc.$), then the output of the dense layer is a new vector in this space, namely $\widehat{WE}_j(t_{input})$, that theoretically should be the closest to the vector that represent the corresponding HPO in the same space. For this reason, the HPO vector closest to the output vector is chosen as synonym of the input sentence, in a similar way to what done in Equation 26:

$$\hat{t} = arg \min_{t_i \in HPO} \left(dist(\widehat{WE}_j(t_{input}), WE_j(t_i)) \right) \tag{27}$$

4.2.3 Convolutional-Encoder-Dense Model

Zhou's C-LSTM model aimed to integrate convolutional models (C-NNs) with R-NNs. Since in his work he was able to improve the performance over LSTM, it could be interesting to test a similar architecture for layman translation. The proposed model, named C-LSTM-D, is reported in Figure 13 and, as in the original C-LSTM model, also this one involves two steps:

1. The **extraction of features** from the input sequence via a convolutional layer.
2. The **acquisition of long-range dependencies** in the feature maps via a LSTM layer.

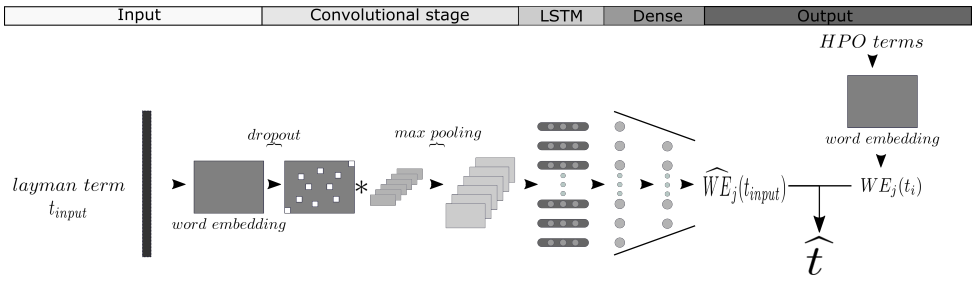


Figure 13: **C-LSTM-D model for layman-HPO translation.**

Even if the C-LSTM-D model is quite similar to the C-LSTM, it is worth highlighting its modifications for the specific task of layman-HPO translation.

The input t_{input} is treated as in the precedent two models, i.e. an **embedding layer** is used to create a matrix $L \in \mathbb{R}^{l_{inp} \times N_{WE}}$ representing the input sentence. A **convolutional filter** is then used to extract features from this matrix and a **max-pooling block** to reduce the dimensionality of the feature space. Since it was proposed in the original C-LSTM model, we added a **dropout** layer before the convolutional step.

The output of the max-pooling layer is still a kind of embedding matrix ($L' \in \mathbb{R}^{l' \times N_{WE}}$, with $l' < l_{inp}$), but now each row represents the most salient features of the words in the embedded space. The second step, i.e. the **LSTM layer**, is then equal to the LSTM-D model: the features extracted by the convolutional layer are mapped by the LSTM in a single vector that is in turn mapped in the HPO embedding by a **fully-connected layer**, obtaining a vector $\widehat{WE}_j(t_{input})$ that is used to predict an HPO term $\hat{t}$ using Equation 27.

4.2.4 Encoder-Parallel Model

As it will be shown in Chapter 6, the current HPO embedding works very well in detecting branches: the WE tends to map HPO terms of the same branch in the same regions of the vector space created. For this reason, even with a simple algorithm as *K-nearest neighbors* one can predict the branch of an HPO term with very high accuracy. Similarly, C-LSTM-D and LSTM-D models, that are created in order to map a sentence in this embedding space, can be used as well to predict the branch of a layman term. The last proposed model, named LSTM-P, is based on the intuition that it will be able to map input sentences to a specific branch in the ontology.

The model is reported in Figure 14 and it is composed by two parallel nets:

- A model for **branch detection** assigns a category to each input.
- A model for **layman translation** assigns a vector in the HPO space to each input.

The idea behind this model it to divide the problem in two sub-problems: first, detect the branch of the input term, in order to reduce the output space in a specific sub-region of the HPO

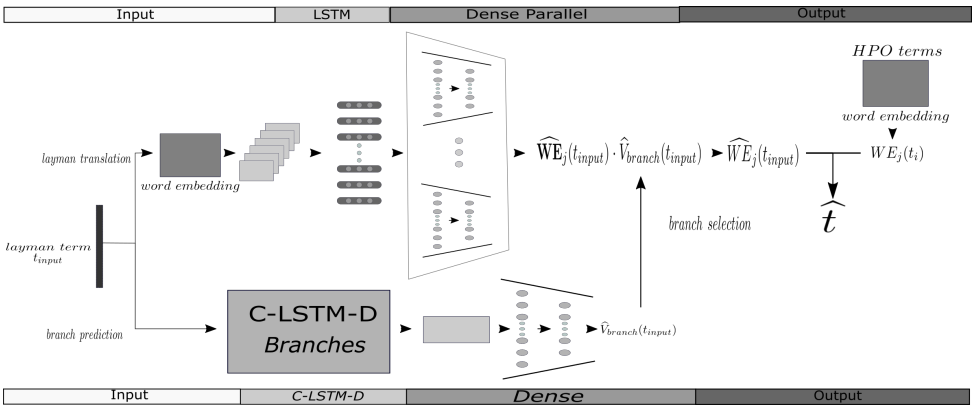


Figure 14: **LSTM-P model for layman-HPO translation.**

embedding; and second, map the term in that sub-region using a model trained specifically to work in that space.

The first model (for branch detection) is very similar to C-LSTM-D, except for the last layer: after the dense layer that maps the output of the LSTM in a vector of the HPO embedding, another dense layer is used to predict the branches, mapping the predicted vector ($\widehat{WE}_j(t_{input})$) in a 26-D one-hot encoded vector ($\widehat{V}_{branch}(t_{input})$) representing the different categories (the 4 sub-ontologies *Mode of inheritance*, *Clinical modifier*, *Clinical course* and *Frequency* were grouped in a unique category). The main reason to use a model based in C-LSTM-D was mainly because it was very fast to train with respect to other alternatives.

On the contrary, the second model is more similar to the LSTM-D model, but with a big difference: after the LSTM chain, now there are 26 parallel dense layers, each one specialized in mapping the input vector in a specific region of the HPO embedding space, according to the predicted branch. The output of this layer is then a matrix ($\widehat{\mathbf{WE}}_j(t_{input}) \in \mathbb{R}^{26\times D}$, with D the dimension of WE_j), the rows of which are different predictions of the input as if it belonged to a specific branch. The correct vector is then picked up using the prediction on branches made by the first model:

$$\widehat{WE}_j(t_{input}) = \widehat{\mathbf{WE}}_j(t_{input}) \cdot \widehat{V}_{branch}(t_{input}) \tag{28}$$

Once obtained a prediction on the embedding of t_{input}, an HPO term $\hat{t}$ is chosen using again Equation 27.

The main advantage of this model is that if a layman term belongs to a specific branch, e.g. *Abnormality of the immune system*, and this is correctly predicted by the C-LSTM-D network for branch prediction, then it will be mapped in the HPO embedding space by a dense layer specifically trained to work with those terms. In this way each branch has a fully-connected layer that maps terms of that branch only in the specific region of space corresponding to them.

5 Experimental design

As mentioned before (Section 3.3), most of the HPO terms contain one or more **synonyms**. In addition, almost all the terms contain a brief **description** of one or more sentences. As a consequence, almost 90% of the HPO terms contain either a synonym or a short sentence as a description, or both. However, a small fraction of HPO will not be represented in the training set since they contain nothing but a label. After a pre-processing step that will be explained further on, this collection of synonyms and descriptions was used as input to the model in order to train and test them. That is, inputs were on one hand layman terms, but on the other hand sentences describing the phenotype. Choosing to include as well the description of the terms in the train and test sets was due mainly for two reasons: first, deep learning models need a huge amount of data and limiting the training set to the layman terms in HPO only would not be enough; second, the introduction of describing sentences made the problem more challenging and ambitious, increasing the range of applicability of the final solution.

More than 16% of the descriptions in HPO are formed by more than one sentence, for this reason descriptions were split in sentences using a sentence tokenizer, and sentences longer than 50 words (less than 1% of the sentences) were discarded. Note that for the encoder-decoder model and for the LSTM-D model the length of the input sentence is not a problem: LSTMs can work with inputs of varying length. On the contrary, CNNs must work with data with fixed dimensionality. For this reason, all the sentences were padded with a special character to be of the same length.

The **pre-processing** algorithm was the same that was used for cleaning the corpus of documents C (see Section 4.1 and Algorithm 1 Appendix A). Besides the classical pre-processing steps, i.e. lower all the uppercase letters and eliminate punctuation, some strides specific for HPO were introduced:

- In HPO ordinal numbers are written with digits, thus they were transformed to strings (e.g. from *Abnormality of the 1st metacarpal* to *Abnormality of the first metacarpal*).
- The presence of couples of numbers separated by a hyphen is quite common, particularly for toe phenotyphes. In these cases, the two numbers were separated and transformed to ordinal numbers (e.g. from *2-3 toe syndactyly* to *second third toe synadactyly*).
- Other numbers were commuted to strings only when they were isolated and not part of a unique word (for example *Tessier number 4 facial cleft* becomes *Tessier number four facial cleft*, but *cervical c3 c4 vertebral fusion* remains equal since the numbers are part of a word).
- Words separated by a hyphen were considered as a unique word, i.e. the hyphen was not removed along with the rest of the punctuation.

Note that, when this algorithm was used for the WE_{LSA}, stop words were eliminated as well and hyphens were treated differently: in order to remain consistent with the Gen. WE, words separated by a hyphen were considered as a unique word only in the case that they belong to a unique string in the McDonald's WE, while they were split in two words in the other cases (e.g. *Stroke-like episode* becomes *strokelike episode*, but from *Multiple non-erupting secondary teeth* one obtains *multiple non erupting secondary teeth* since *non-erupting* was not an item of the McDonald's embedding).

Obviously, it would be impossible to define a set of pre-processing rules that could work for each particular case in each sentence: the rules stated above work for almost all the cases and thanks to the huge amount of input data, the exceptions will not influence the results. All the pre-processing steps were done using the Natural Language Toolkit (NLTK) [61].

After preprocessing, a list of **30,230 layman terms or sentences** was created. This list contains terms that represent almost 90% of HPO terms but, as mentioned before, a small fraction of phenotypes is not represented in the train set. Furthermore, 33% of the HPO classes are represented only by a layman term or a sentence, while 24% of the HPO classes is represented by two layman terms or sentences. The remaining 33% of the classes is represented by 3 or more terms, with some classes with more than 10 layman terms or sentences and a mean of 2 descriptors per term. Further details about the train set will be given in Section 6.1.

This work was focused in testing the models in combination with the 5 embeddings proposed in Section 4.1. During the first phases of the experiments, the encoder-decoder model appeared to be very costly in terms of RAM utilization and training time. Since the first results obtained appeared to be worse in comparison with the other architectures, this model was discarded. Nevertheless, to define it was a crucial step since the other models are an evolution of this first one.

As mentioned before, in McDonald's WE there were 2 models, one of dimension 200 and one of dimension 400. Both models were tested in order to analyze how the output dimension influences the results. Theoretically, for WE_{LSA} and WE_{SVD} the dimension can be any integer number. In order to be consisten with the Gen. WEs only WEs of dimensions 200 and 400 were tested. Further on, the dimension of the WE will be referred with letters A and B respectively, thus $WE_{G1\,A}(t_i)$ will denote a vector of size 200 obtained with the first general embedding technique and $WE_{G1\,B}(t_i)$ will denote a vector of dimension 400 obtained with the same embedding technique.

In addition to the 5 embeddings, also a random WE ($WE_{rand}(t_i)$) was created. Each feature was randomly generated using a different Gaussian mixture model estimated from the corresponding features of WE_{G1}.

Each model has a different architecture and hence different tunable parameters, however some are common among the architectures and their influence in the results can be investigated. In particular, the initial embedding layer and the LSTM layer appear in all the models: different dimensions of these two layers were tested.
Referring to the C-LSTM-D model, after the first experiments it seemed that the model did not suffer from over-fitting problems, hence the dropout layer was removed.
Details on the parameters are reported in Table 12 in Appendix B.

The development of this project was totally carried out with Python: the code to creates the models and to pre-process the test can be found in Appendix D . The models are trained with cross-validation, with 29,625 terms on the training set and 605 on the test set. The model is built

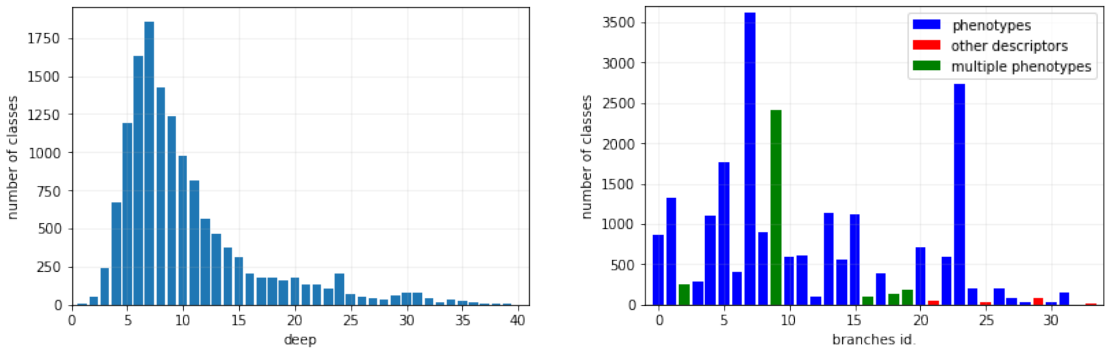
and trained on Keras [62] with mean squared error as loss function and the Adam optimization algorithm.

6 Results

In this chapter, results of the model described in Chapter 4.2 will be analyzed. Furthermore, the impact of the different WEs introduced in Chapter 4.1 will be studied, considering on one hand how they affect the performance of the model and, on the other hand, investigating the vector space created. However, without knowledge on how the training set is distributed in HPO, the subsequent results could be misunderstood and misinterpreted. For this reason the first section of this chapter is dedicated to the analysis of the frequency of single words in test set and HPO. The second section is then focused in analyzing how the WEs proposed worked, and in the third part the final results of the complete models are reported and studied.

6.1 Statistical description of HPO words distribution

In Chapter 2 HPO was briefly described and two important concepts related to HPO terms were introduced: the depth of a term defined as the number of hypernyms that the term itself has; and the branch or category of a term, defined as the direct descendants of the *Phenotypic abnormality* node (plus other 4 describing nodes). Moreover, in Chapter 5, it was explained how the classes in HPO were used to create a training set for the models. The aim of this section is to further analyze this training set, considering in particular the distribution of HPO classes and single words among the branches and the depth in the ontology.



(a) *Number of classes per deep*

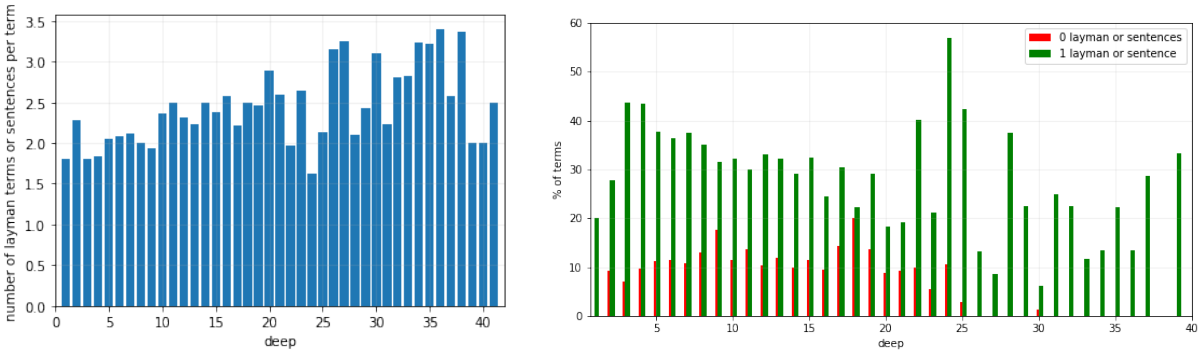
(b) *Number of classes per branch*

Figure 15: **Distribution of classes inside HPO.**

In Figure 15, one can note the distribution of the terms inside HPO, with respect to the depth of the terms (15.a) and to the branch (15.b). Note that, for graphical reasons, branches are indicated with an identification number: the corresponding branches for each `id` are reported in Table 13 in Appendix B. The blue columns in the *branches vs # of classes* histogram indicates phenotype branches, i.e. the branches that are direct descendants of the *Phenotypic abnormality* class. As expected, almost all the classes are included in one of these branches. Nevertheless, the distribution of terms among these branches is not uniform, with some branches with more than 1,000 terms (e.g *Abnormality of the skeletal system* corresponding to the 7th column, or *Abnormality of limbs*, corresponding to the 23rd column) and other branches that refer to more specific abnormalities and hence contain only a few thousand terms or even less, as *Growth abnormality* or *Abnormality of the voice*, corresponding to columns 12 and 30, respectively. On the

contrary, the red columns in the same histogram correspond to terms in one of the four descriptor classes (*Mode of inheritance*, *Clinical modifier*, *Clinical course*, *Frequency*). The number of terms they provide to the ontology is very limited, but this is not a big issue: since the aim of this work is to identify phenotypes, it is of greater importance for the other branches to be well represented, while these categories are less relevant. Finally, there is a third type of column: the green columns represent terms belonging to more than a branch [1], e.g. column 9 contains terms that simultaneously belong to *Abnormality of the skeletal system* and to *Abnormality of limbs*. Not every possible combination is reported in the histogram, but only the ones that represent more than 100 terms. As one could imagine, these categories that partially overlap are somehow connected by semantic similarity, as in *Abnormality of the skeletal system-Abnormality of limbs*, or in *Abnormality of blood and blood-forming tissues-Abnormality of the immune system*.

The first histogram, that counts the number of classes at each depth level in the ontology, shows that most of the terms are located between a depth of 5 and a depth of 10. These are usually terms that are quite specific and already identify a phenotype, for example *Pulmonary edema* ($depth = 8$) or *Intermittent hyperpnea at rest* ($depth = 7$) as opposed to the term at shallower level of depth, that usually indicates more a family of phenotypes (e.g. *Abnormality of the musculature of the thorax* ($depth = 3$)) rather than a specific phenotype. As the depth level increases, the specificity of the terms also increases, with phenotypes usually involving very specific parts of the body like phalanges or small bones of the metacarpals. From a point of view of the task of this work, the most useful terms are the ones that describe specific phenotypes and can be found in the middle part of the histogram and in the tail, approximately from a depth level of 7/8. Consider a real application in which patients (or even clinicians) use an application based on this work to find an HPO term based on their symptoms: it is legitimate to think that their will be more interested in finding out the name of the specific phenotype that affects them, rather than getting a generic description of phenotypes affecting a part of the body as *Abnormality of ...*



(a) *Number of entries in the training set per term per depth*

(b) *% of term with 0 or 1 entries in the training set.*

Figure 16: **Distribution of entries in the training set inside HPO.**

[1] In the histogram terms in more branches are counted one time for each branch they belong plus one time for the corresponding multi-branches column.

The main drawback of any deep learning model is the huge amount of data that is needed for a successful training phase. The training set created in Chapter 5 is considerable, containing more than 30,000 entries. However, it is also important how the terms in HPO are represented by this training set. In Figure 16.a an histogram of the number of entries in the training set with respect to the number of classes at a specific depth is reported. As it is possible to note most of the classes have two corresponding items in the training set (the classical situation is in fact that a term contains at least one synonym and a descriptive sentence), even if this is not a strict rule and a term can have both more synonyms or describing sentences, or less. In Figure 16.b the percentage of terms without representation in the training set (red histogram) and with only one representation (green histogram) is shown. Inside HPO, approximately 10% of the terms is not represented in the training set: one can note that these terms are grouped in the upper levels of the ontology and starting from a level of 25 almost all the terms are represented. From a point of view of the layman translation task, it is important that specific terms located at the lower levels of the ontology are well represented, while it is less relevant that some terms in the upper levels are not represented.

The fact that more specific concepts appear to be over represented in terms of descriptions and synonyms inside HPO can be noted also looking at the trends of the two histograms: the number of entries per term appears to grow with the depth, while the percentage of terms with just one entry tends to decrease as depth increases. It is worth spending some words about the apparently singularity around level 24 where the number of entries per term reaches its minimum and the percentage of terms with one entry has a peak. Of the 207 terms located at a depth of 24, 150 are terms related with the *epiphysis* (i.e. the ends of long bones). Most of these terms contain no description and only a synonym that talks about *end part* rather than *epiphysis* explaining the peak in the second histogram. However, excluding these phenotypes regarding a particular finger abnormality, the trends are confirmed.

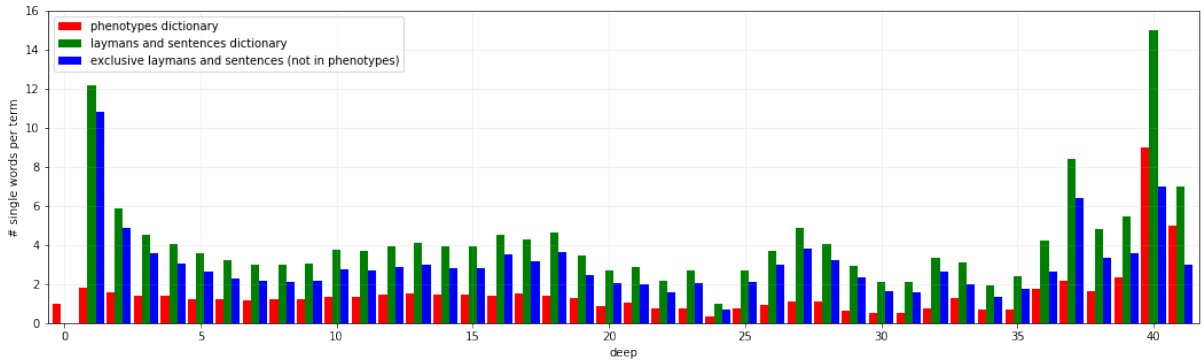


Figure 17: **Distribution of words per term at level of depth**

Aside from the number of synonyms or sentences that an HPO term has, it is important to evaluate the morphological information that these synonyms bring. It is different to correctly identify a synonym that is very similar to the corresponding HPO term and to identify a synonym that uses totally different words. For example, the HPO term *Increased circulating chylomicron concentration* has three synonyms, two of them (*Increased chylomicrons* and *Increased circulating chylomicron levels*) do not provide much additional information to the term: it could be said that they only shuffle the words of the original label. On the contrary, the third synonym

(*Hyperchylomicronemia*) is totally different from the original one. In figure 17, three histograms are shown, displaying:

- In red, the number of words in the HPO dictionary per depth level (normalized by the number of HPO terms).
- In green, the number of words in the training set dictionary per HPO term, i.e. the number of words in the dictionary of the synonyms and sentences with respect to the number of HPO classes at that depth.
- In blue, the number of words in the training set that do not belong to the dictionary of HPO classes, i.e. the new words that synonyms and sentences contribute. If A is the set of words in HPO dictionary and B the set of words in the training dictionary at a certain depth, then this third set is $C = B - (A \cap B)$ (see Figure 18)

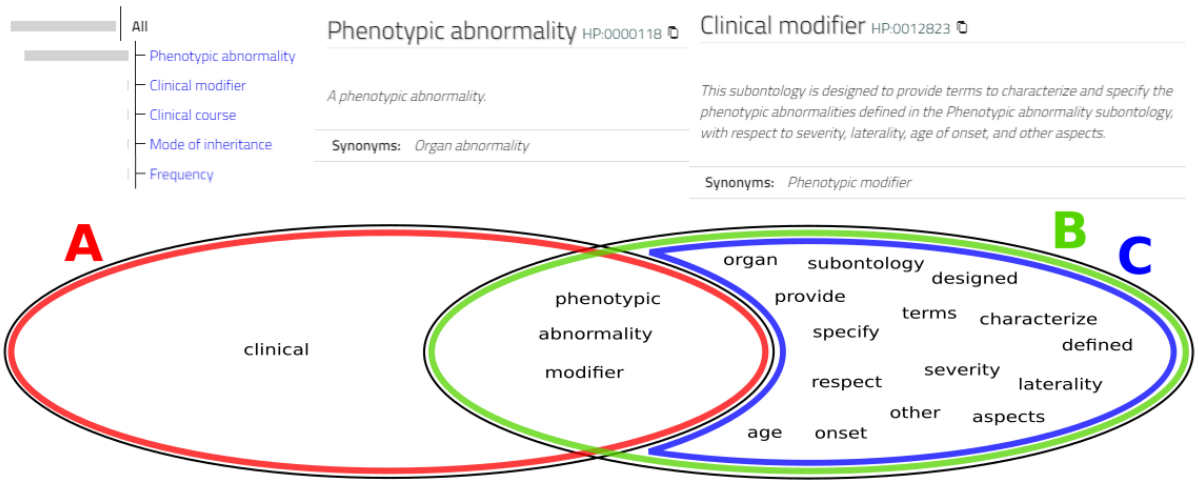


Figure 18: **Example of words distribution at level 1.**

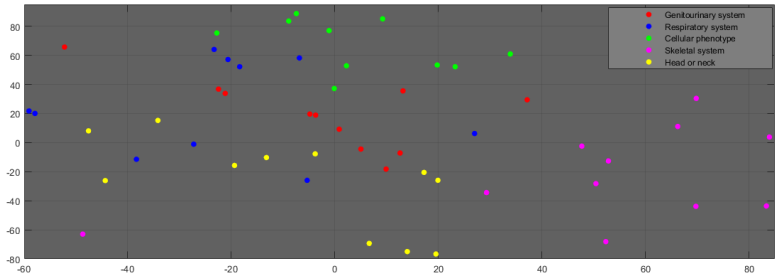
The number of words in each of the three histograms remains more or less constant among levels: the peak at the lower levels is due to the fact that these are very specific terms, with long sentences that define them (as *Symphalangism of the proximal phalanx of the 2nd finger with the 2nd metacarpal*). Note that at a depth of 24, the morphological variation is limited, due to the repetition of terms as *epiphysis* or *finger*.

The same statistics, i.e. the distribution of layman terms and sentences per term and the number of words per term, can be calculated also according to the branches. However, inside a branch there is more variety than inside a depth level, i.e. even if terms of the same branch are related by a kind of semantic relationship, they are very different among them in terms of specificity, number and type of words and so on. For these reasons, the same histograms of Figures 16 and 17 but calculated per branch instead of per depth are less informative. The results are more uniform along categories and branches are not significantly different. Furthermore, except for the four descriptive branches that will be unified in a unique category in further analysis, no branch is more relevant than others from a point of view of the layman translation

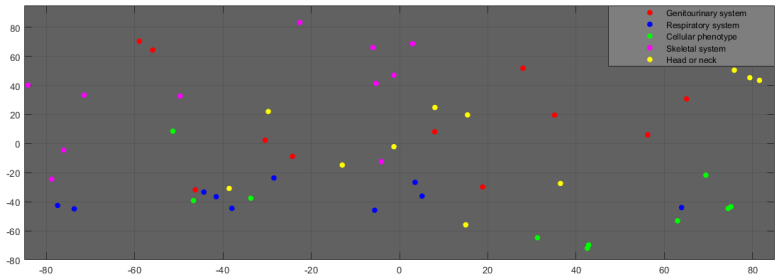
task.[2]

6.2 Embeddings results

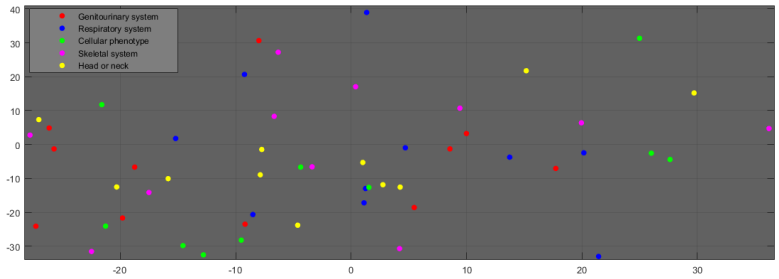
The evaluation of an embedding is not an easy task. There is no way to determine the most suitable vector representation of a word or a concept . Since each method has its pros and cons, and the evaluation of the effectiveness of a WE must be done considering the task at hand.



(a) $WE_{G1\ A}$



(b) $WE_{LSA\ A}$

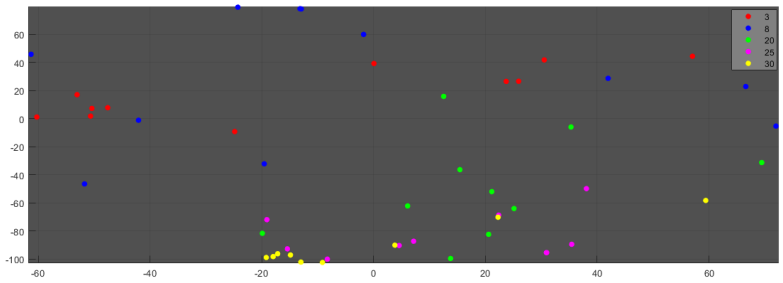


(c) $WE_{Rand\ A}$

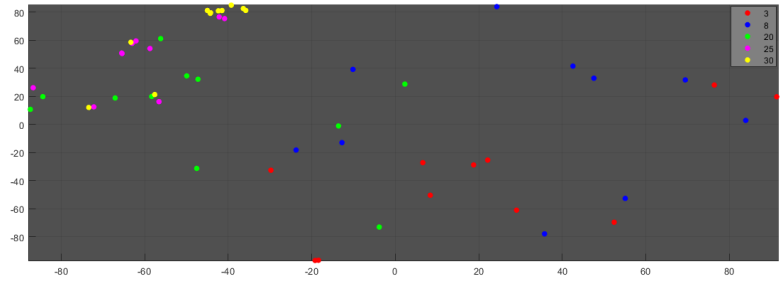
Figure 19: **WEs representation in a lower dimension space** (stratified by branch).

Intuitively, a WE should map semantically related concepts in vectors close in the WE space; for example two concepts as *Vascular calcification* and *Vascular tortuosity* should be represented

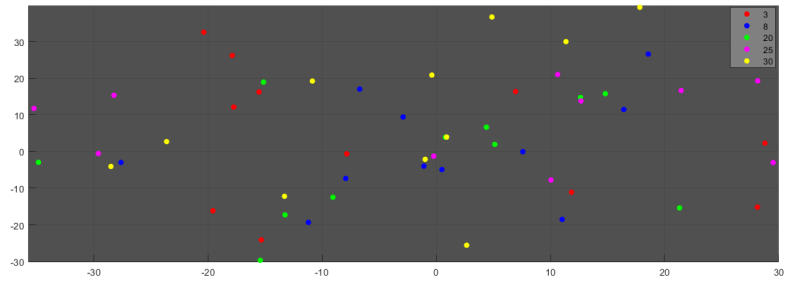
[2] A visual representation about the distribution of words and layman terms inside each branch is reported in the histograms in Appendix C, Figure 31 and Figure 32.



(a) $WE_{G1\,A}$



(b) $WE_{LSA\,A}$



(c) $WE_{Rand\,A}$

Figure 20: **WEs representation in a lower dimension space** (stratified by depth level).

by two vectors very close, while a third concept e.g. *Short metatarsal*, should be mapped in a completely different region of the space. However, this objective is not straightforward: for the specific case of HPO, there should be more then 13,000 vectors in $\mathbb{R}^{200}$ or in $\mathbb{R}^{400}$. A visual validation of the WEs is shown in Figures 19 and 20. Using the Hypertools [63] toolbox for data reduction and visualization, some terms were plotted in a 2D using PCA dimension reduction [64] to reduce data from $\mathbb{R}^{200}$ to $\mathbb{R}^{50}$ and then using t-SNE algorithm to visualize them in 2D [65]. In particular in Figure 19, 5 branches were selected (*Abnormality of the genitourinary system*, *Abnormality of the respiratory system*, *Abnormal cellular phenotype*, *Abnormality of the skeletal system* and *Abnormality of head or neck*) and for each branch 10 phenotypes were randomly picked up. Each term was represented in three different spaces: $WE_{g1\,A}$, $WE_{LSA\,A}$ and WE_{rand} to have a reference. Figure 20 was obtained with the same procedure, but the terms were selected from different depth levels. These graphical representations are limited: figures display a 3D representation of vectors in $\mathbb{R}^{200}$ and hence they can not capture the complexity of the space. However, even if they are simplified representations, some observation can be carried out. In

Figure 19.a , the branches appear to form clusters in the space. The green points, corresponding to *Abnormal cellular phenotype* branch, for example, are separated from the other points and clustered in the North region. The same is valid for the purple points (representing *Abnormality of the skeletal system*) that occupy the East region. The fact that terms of the same category tend to cluster is also evident in Figure 19.b, even if in this case the trend is less evident. It is interesting to notice that the points corresponding to *Abnormal cellular phenotype*, a branch that is semantically distinct from the other four, appear to be separated from the other points in both cases. Finally, the random WE appears as expected: points are scattered in the space without structure. Figure 20 is less informative. At first glance, in both embeddings (Fig. 20.a and 20.b) it seems that more generic terms, i.e. terms at a lower depth (corresponding to red and blue points), tend to concentrate in a region of space different from the one of the other points. However, there are points that do not respect this rule and the differences between these two embeddings and the random one are less pronounced.

The intuition that terms in HPO tend to create clusters inside the WE space according to the branches they belong can be tested using the k-NN algorithm. The basic idea of this classification algorithm is classifying an element in a feature space according to the most common class among its k nearest neighbors in the feature space [66]. In this case, 1,000 HPO classes were randomly selected and, for each one, a prediction of the branch was carried out based on the branches of the $k = 15$ nearest terms in the WE space. Since many terms belong to more than one branch, the classification was considered correct if at least one of the branches was correctly classified. Numerical results reported in Table 1.a confirm the hypothesis: all WE correctly classify the terms in almost 90% of the cases .

WE	Correct classification [%]	
	A	B
g1	89.7	89.0
g2	**90.4**	**90.9**
g3	87.7	88.5
lsa	87.7	**90.9**
svd	89.5	89.1
rand	22.2	22.1

(a) Branch prediction

WE	Correct classification [%]	
	A	B
g1	55.5	56.3
g2	56.3	57.0
g3	52.9	52.0
lsa	**59.6**	**62.0**
svd	55.9	56.3
rand	30.2	29.2

(b) Depth prediction

Table 1: **Percentage of terms correctly classified with K-NN.**

The same experiment was performed also to try to classify a term according to its depth. As before, k-NN was used to detect the depth of 1,000 random classes. In this case, a prediction was considered correct in a range of ± 1 around the correct depth (e.g. if the correct level was 5, then a correct prediction could be in the interval [4-6]). Results in Table 1.b are consistent with the visual intuition from Fig. 20: the embeddings can not represent depth as well as they represent branches.

Another property that should be checked to evaluate an embedding in the specific case of HPO is that it should keep the relationship among terms unchanged, i.e. terms with a high

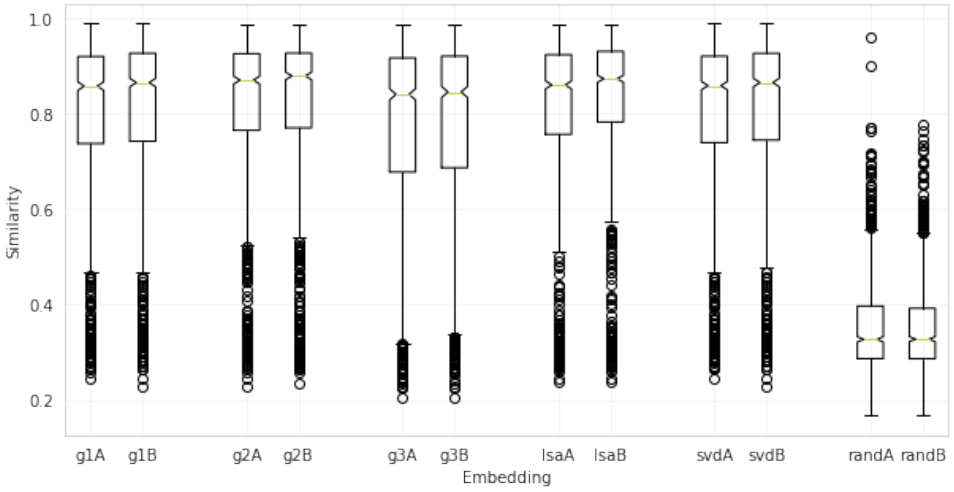


Figure 21: **Similarity between HPO terms and the closest vector in the corresponding WE.**

similarity in the ontology should be mapped close in the WE and the other way around, pairs of vectors close or far from each other in the WE should represent couples of terms with a high or low similarity in HPO. Figure 21 shows a boxplot representing each one of the WEs proposed in Chapter 4.1. The boxes are calculated picking up random HPO classes and calculating the similarity between them and the HPO term that corresponds to the closest vector in the embedding, i.e. given an embedding WE_j and n terms $t_i, i = 1, .., n$, each box is obtained calculating:

$$s_i = sim\big(t_i, arg \min_{\tau \in HPO} dist(WE_j(t_i), WE_j(\tau)\big) \quad i = 1, .., n \tag{29}$$

Intuitively, close points in the vector spaces should represent close concepts in HPO, thus the similarities calculated between random pairs of points close among them should tend to one. Mean and standard deviation of the similarity among the WEs are reported in Table 14 in Appendix B. Again, results shows that all the embeddings have a mean similarity between 0.76 and 0.82, indicating that the relationships among terms is in that way preserved. Furthermore, results are aligned with the ones in Table 1, with two WEs (WE_{g2} and WE_{lsa}) that appear to perform slightly better than the others, and with the WEs in $\mathbb{R}^{400}$ that in general work marginally better. However, the standard deviation is high for all the WEs as it is the number of outliers, indicating that, even if these feature spaces can work in general, they are still rudimentary and they can not represent all the terms correctly. The last observation is that WE_{g3}, that theoretically should be the best one among the Gen. WE since it was created with more words related to the single concepts, in practice is the one that works the worst. The highest standard deviation of this embedding shows how the list of words used to create it is more a source of noise rather than an improvement of the representation.

Another way to evaluate the embeddings is to map the layman terms directly in this space and try to predict the corresponding HPO class. Given a layman term t_{input}, it is possible to get its representation in one of the embedding spaces ($WE_j(t_{input})$) applying the same steps that were applied at the HPO terms to get the corresponding WE. Hence the closest vector in the WE space can be chosen as synonym the input:

$$\widehat{t} = arg \min_{\tau \in HPO} dist(WE_j(t_{input}), WE_j(\tau)) \tag{30}$$

In Table 2 some numerical results are reported. The prediction of a layman is evaluated in terms of similarity between the predicted HPO class $\widehat{t}$ and the correct one t_c. The similarity of a pair of terms varies between 0 and 1, and it is not the same if $\widehat{t}$ is different from t_c but very similar, or if it is completely different. For these reasons, predictions were divided in three cases: a prediction was considered **correct** if the similarity $sim(\widehat{t}, t_c)$ was equal to 1, while it was considered **wrong** if $sim(\widehat{t}, t_c) \leq 0.7$. In the case $1 \leq sim(\widehat{t}, t_c) < 0.7$ the prediction was defined **near**. To get an idea of the level of similarity between terms consider for example that two terms as t_1 =*Sparse scalp hair* and t_2=*Slow-growing scalp hair* have a similarity $sim(t_1, t_2) = 0.82$, hence they would be classified as near, while the similarity between t_2 and t_3 =*Difficulty adjusting from dark to light* is $sim(t_2, t_3) = 0.29$, and hence this classification would be considered wrong. Furthermore, the rank of the correct term in the sorted list of most similar vectors in WEs space was calculated. For each embedding, the median of the ranks is reported. Finally, the last column of the table reports the percentage of layman terms for which the predicted HPO term was in the same branch of the correct HPO term, regardless of the correctness of the prediction.

	Correct [%]		Near [%]		Wrong [%]		Median rank		Correct branches [%]	
WE	A	B	A	B	A	B	A	B	A	B
g1	20.8	21.2	62.9	64.5	37.1	35.5	17.0	16.0	89.8	89.6
g2	**27.2**	**27.8**	**73.1**	**74.2**	**26.9**	**25.8**	**6.0**	**5.0**	**90.8**	**90.7**
g3	12.5	13.1	54.5	56.2	45.5	43.8	63.0	51.0	90.3	90.5
rand	0.0	0.0	1.5	1.0	98.5	99.0	6920.0	6871.0	18.2	17.0

Table 2: **Results of the WEs used to predict layman terms.**

Note that, unlike the Gen. WEs, in the embedding WE_{lsa} each vector was a representation of the whole HPO class considered as a document. That means that each synonym in this embedding has the same representation of the corresponding term, making this test of little relevance to evaluate it. For this reason in Table 2, only the Gen. WEs were tested (plus the random one as reference). Furthermore, this test was carried out only on the layman terms in the training set, and not on the sentences. In fact, all the proposed WEs are thought to represent concepts of few words (between 2 and 4 according to histogram in Figure 17). The Gen. WEs in particular were created as sum of vectors: it would be meaningless to embed sentences of dozens of words using these techniques.

Numerical results of this test confirm what has been discussed so far: in comparison with the random WE, WEs appear to be able to represent the ontology in a vector space; WE_{g2} is the embedding that has best results, while WE_{g3} is the worst. In all three cases, the branches are correctly spotted in almost all the terms.

Another aspect that could be interesting to investigate is the effectiveness of these embeddings encoding the depth of the input terms. In Figure 22 the base-10 logarithm of the median rank per depth level is reported. As expected, except for some cases, WE_{g2} (Fig. 22, blue histogram) has the lower value at each level. However, the most interesting thing to note is the similar trend that all three WEs have: at lower levels the embeddings appear to work very well, while the

median rank explodes as the terms become more specific (i.e. the depth increases). A possible explanation of this fact is that as the terms become more specific, the vocabulary to describe it is also more complex and the embedding base of the three Gen. WEs can not describe these words as well as the more generic ones at lower levels.

6.3 Machine translation results

As reported in Table 12, Appendix B, there are plenty of tunable parameters for each model. Furthermore, each model should be tested with each proposed WE, reaching a high number of possible combinations. Just considering the number of parameters that have been chosen for tuning (that are only a part of the total number of parameters), the total number of models that should be tested is $N = numb.\ of\ models \times number\ of\ WEs \times dimensions\ WEs \times dimensions\ input\ WE \times dimensions\ LSTM = 3 \times 6 \times 2 \times 2 \times 2 = 144$, i.e. each model has 48 possible combinations. The number of trainable parameters varies as a function of the model, from a minimum of $\sim 7M$ tunable parameters to a maximum of 24M. In particular, LSTM-P is composed by two parallel models, and hence the number of parameters is almost twofold with respect to the other two models. Accurate dimensions of the three models are reported in Table 15 in Appendix B, along with the time needed to train the models[3].

This huge number of trainable parameters has an impact on the time needed to train the models, but the characteristic that influenced the training time the most was the presence of a convolutional layer. R-NN are very slow to train, one explanation can be found in the dependencies of each time step of the R-NN on the previous steps that limits the possibility to make calculations in parallel across timesteps and batches [67]. Furthermore, usually GPUs are built with a special focus on convolution and C-NN rather than R-NN, due to the fact that computer vision and deep learning techniques mostly employ CNN. To conclude, the convolutional layer, even if it implies additional parameters, decreases the input size of the LSTM layer, reducing the training time 4/5 times.

[3]Time to train depends on several factors: the GPU, if someone else was using the CPU in parallel, the models, etc. Reported times are only an approximation based on empirical observations.

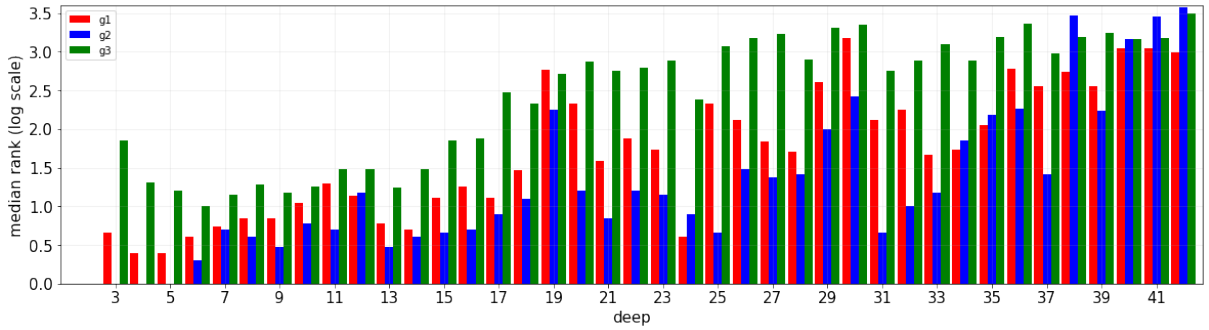


Figure 22: **Median rank (in logarithmic scale) at different depth levels for the three Gen. WEs.**

The total time needed to cross-validate all the 144 combinations of models was of approximately 60 days, with a CPU completely dedicated to these calculations 24 hours per day. Obviously, it would be impossible to obtain all these data by the conclusion of this project. For this reason, the complete combination of tunable parameters was tested only for the fastest C-LSTM-D model, while the other two models have only partial results.

The notation used to refer to a specific model will be the one proposed in Table 16, i.e. :

$$MODEL(WE_j, Embedding\ \ layer\ \ dim., LSTM\ \ dim.) \tag{31}$$

6.3.1 C-LSTM-D model results

Particular attention is given to the analysis of the C-LSTM-D model results, since they are the most complete ones. The analysis of the outcomes is carried out through observations of branches, depth levels, and similarity, used as main metrics to evaluate the results. All the numerical results are reported in Table 17, Appendix B. The definition of **correct**, **near** and **wrong** is the same as before. Even if all the information, including branches and depth classification, is contained in this table, it could be useful to report some additional statistics, in order to better visualize the behaviors of the different models.

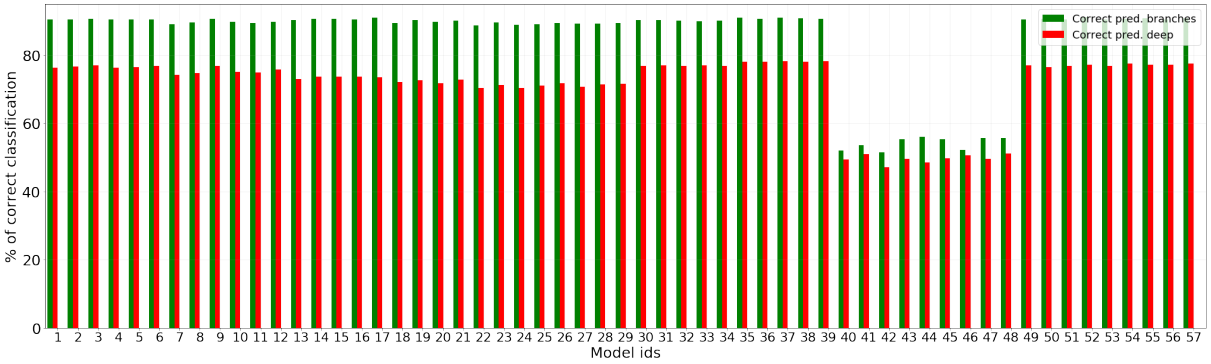


Figure 23: **Percentage of correct classification for branches (green) and depth levels (red) per model. (C-LSTM-D)** (`Ids` of the models are reported in Table 16, Appendix B)

In Figure 23, the percentage of terms correctly classified according to their branch or level of depth is reported. The models are identified by a numerical `id`, according to Table 16 in Appendix B. In order to be consistent with the results of the WEs, in the cases of terms belonging to more than a branch, a prediction was considered correct if at least one predicted category was correctly classified. Similarly for depth, the prediction was considered correct within an interval of ± 1 around the correct depth level. For branches classification, results are similar to the one obtained with the WEs. However, it is interesting to note that the differences between WE_{g2} and WE_{g1} are more flattened, i.e. there is not as much difference as there was using only the embeddings. On the contrary, WE_{g3} performs worse with respect to the other embeddings. The best models are the numbers 52 and 35, corresponding to C-LSTM-D($WE_{lsa\ B}$,400,400) and C-LSTM-D($WE_{svd\ A}$,600,600). Nevertheless, differences are very limited among different models.

On the contrary, predictions on the depth level are different from the ones obtained with the usage of only the WEs: if before it was demonstrated that only using the WEs was not enough to represent terms according to their specificity, now the introduction of a neural model improves performances drastically, reaching almost 80% of correctly classified samples for various models. Again, the best models are the ones based on the DS embedding and the combined WE.

The random embedding based models are probably the most informative ones: for both branch and depth classification all the models reach percentages around 50%. Considering that these values are twice the ones obtained using only WEs, the improved performance and effectiveness of the C-LSTM-D model is confirmed. To sum up, the model appears to be able to identify with good precision both the categories and the specificity (i.e. the depth) of the input terms.

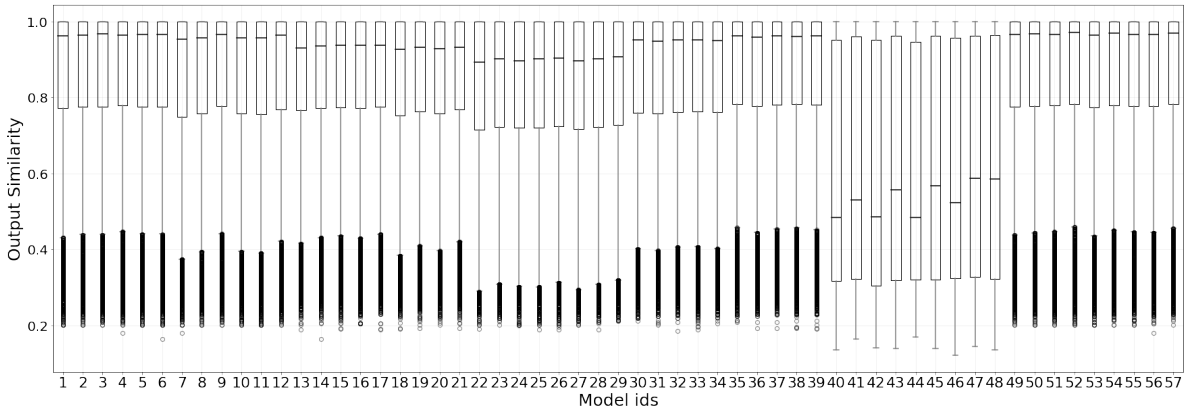


Figure 24: **Predicted similarity per model.** (`ids` used to represent the models are the ones of Table 16)

Figure 24 contains a boxplot of the similarities between predicted HPO classes and corresponding correct HPO classes. Even if variations among the different models are marginal, some comments can be done, especially on the fluctuations detectable between embeddings. Looking at the results in Table 17, one can note that the first Gen. WE performs in general better than the other two. It is unexpected, especially considering that all the tests on the WEs suggested that it is the second Gen. embedding the one that better represents HPO. A possible explanation could be that the vectors in WE_{g2} are closer among them since they are calculated with a weighted sum, and the model prefers to work with more spaced vectors. Results of the DS WE and of the combined one are comparable to the ones of WE_{g1}, even if the dispersion appears to be lower, especially for the WE_{lsa} models: for example the best model in terms of mean similarity based on LSA (C-LSTM-D($WE_{lsa\ B}, 600, 600$)) has a standard deviation 5% below the one of the best Gen. embeddings one (C-LSTM-D($WE_{g1\ A}, 400, 600$)). However, the scatter of the similarity prediction appears to be the biggest issue of the C-LSTM-D model. Finally, the combined embedding WE_{svd} outperforms both Gen. and DS embeddings, confirming the it can capture the strengths of the two WEs.

An interesting issue is the dimensionality of the output space: surprisingly, the Gen. WEs based models work better when the output space is $\mathbb{R}^{200}$ instead of $\mathbb{R}^{400}$, contradicting the re-

sults in Table 2, Section 6.2. On the contrary, when the output space is based on LSA, the model performs better in $\mathbb{R}^{400}$ rather than in $\mathbb{R}^{200}$. This last result is more logical: since LSA is obtained via a SVD dimension reduction, the embedding in $\mathbb{R}^{200}$ is a compressed version of $\mathbb{R}^{400}$, and hence it should convey less information. Finally, the combined embedding is aligned with the Gen. WEs, i.e. models in $\mathbb{R}^{200}$ work better than models in $\mathbb{R}^{400}$.

The last parameters that should be analyzed, the dimensions of the embedding and of the LSTM layers, do influence the results: if one considers the number of terms correctly classified, the number of *near* classification or the median of the predicted similarity, for almost all the embeddings the best combination appears to be C-LSTM-D($WE_j, 600, 600$). That means that using wider layers may have a positive influence on the models, without over-fitting the input data.

All the combinations were tested using two different statistical tests:

- The **Mann–Whitney U test**, a non-parametric test with null hypothesis H0 ($p > \alpha$) stating that a randomly selected value from one population has the same probability to be less than or greater than a randomly selected value from the second population [68].
- The **Kruskal–Wallis test**, a non-parametric test with null hypothesis H0 ($p > \alpha$) stating that the medians of two or more populations are equal [69].

Vectors containing predicted similarities from two models were used to test if the two combinations can be considered different or not. For both tests, all the experiments rejected the null hypothesis when one of the two models included the random embedding, confirming that a structured HPO embedding actually has an influence in the results. The dimensionality of the output space also has a statistically influence in the results, even if not in all the cases. In general, it was noted that for higher input embedding and LSTM dimensionality, the HPO WE does not influence the outcomes, while if there were lower, the influence was more evident: for example considers lines 1 and 2 in Table 3 against line 3.

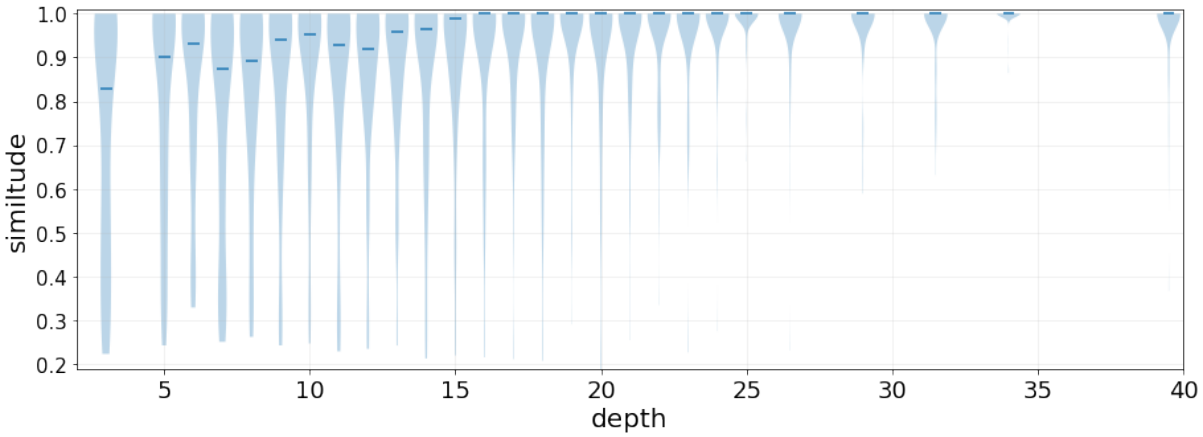
Models compared	p value (Mann–Whitney U, $\alpha = 0.05$)	p value (Kruskal-Wallis, $\alpha = 0.05$)
(WE_{g1A} 600 600) vs (WE_{g1B} 600 600)	0.075	0.15
(WE_{lsaA} 600 500) vs (WE_{lsaB} 600 500)	0.99	0.0011
(WE_{g1A} 400 400) vs (WE_{g1B} 400 400)	8.14e-05	0.00016
(WE_{g1A} 400 600) vs (WE_{lsaB} 600 600)	0.227	0.453
(WE_{g1A} 600 600) vs (WE_{g3B} 600 600)	1.2e-74	2.5e-74

Table 3: **Results of Mann-Whitney U test and Kruskal-Wallis test for different models.**

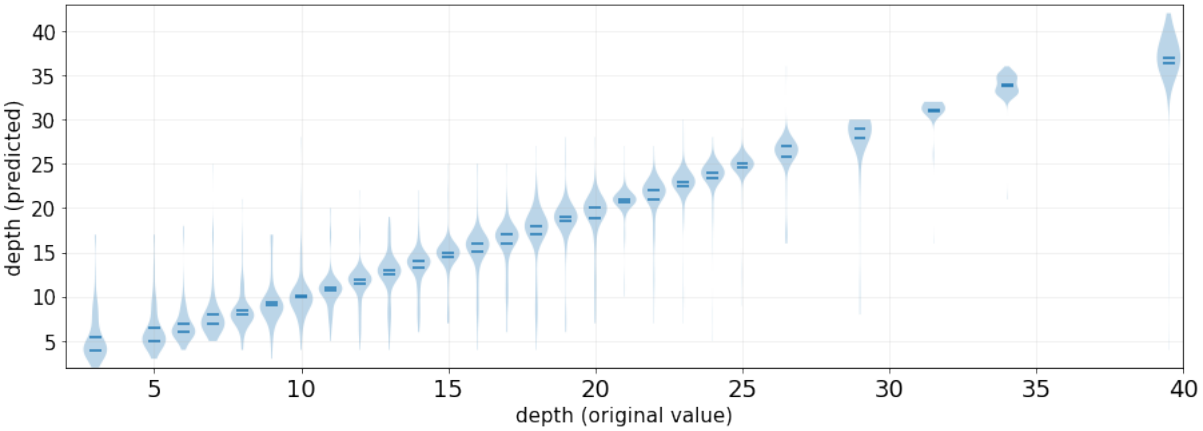
To conclude, irrespective of the HPO embedding or the combination of parameters, in general there were no statistically significant differences between the models that better worked in terms on median similarity or percentage of correctness (consider for example line 4 on Table 3 were two high performances models were compared). On the contrary, and this is the most important factor, the distributions and the medians are different among models that perform well and models that do not (an example is in the last line of Table 3).

I also analyzed if there are terms that are always misclassified, independently from the model. For example there are 195 terms that are always misclassified: considering that the mean of misclassified terms for the models (excluding random models) is 2536, it corresponds to 7.7% of the errors. This percentage quadruples when one considers as wrong the terms with a predicted similarity $sim(\widehat{t}, t_c) \leq 0.5$, reaching 27.4% of misclassified term for all the models. That means that there is a set of terms that is always misclassified with a big error on the prediction. These terms are often synonyms morphologically unrrelated to the corresponding HPO classes, and with few similar terms in the training set (e.g *Wasting syndrome* for *Cachexia*, or *Fainting spell* for *Syncope*). Probably, with a bigger training set that includes these expressions, these errors would have a smaller impact.

The effects of the different tunable parameters have been extensively analyzed in the precedent pages; however, the behavior of the C-LSTM-D model with respect to depth and branches is still to investigate. For this reason, model C-LSTM-D($WE_{g1\,A}$, 600, 400) will be used to study its performance. This model was chosen since it was one of the top models, but not the best ones(Correct predictions: 47.74%; Near predictions: 80.55%; Median similarity: 0.966; Correct branch predictions: 89.06%; Correct depth predictions: 74.24%).



(a) Predicted similarity per depth level.



(b) Predicted depth level per depth level.

Figure 25: **Results per deep for the testing model.**

Figure 25.a contains a graph of the similarity between the predicted output and the correct one at different depth levels; Figure 25.b instead represents the depth level of the predicted output per levels of depth (of the correct output). Each *violin* represents 100 random samples. Some levels that did not contain enough points were merged together (the last 10/15 levels were grouped in 5 groups).

On the contrary to what was observed in the case of the WEs, that worked well for unspecific terms and worse with increasing depth(see Fig. 22), this model performs better as the specificity of the terms increases. One can note that the median similarity (indicated with a blue dash) tends to increase with increasing depth, starting from a minimum of 0.88 for the upper levels and saturating at the maximum value of 1 after level 15. Furthermore, also the dispersion and the outliers, that are some of the biggest limitations of the model, appear to be concentrated on the upper levels of HPO, and they diminish on deeper levels. Since the HPO embedding spaces alone behaved in the opposite way, it is presumable that these results are due to the action of the model. Furthermore, comparing this image with the one obtained with a random WE (Figure 33.d in Appendix C), it appears evident that an ordered embedding space is crucial for a correct translation and the model alone is not able to spot the correct HPO class, whether general or specific. Nevertheless, from a point of view of the main objective of this work, i.e. a machine to help patients identify their phenotypes, this behavior is promising. It is expected that the more specific a term is, the most probable is that a patient does not know it: for this reason it is more important that the model works better with more specific terms, rather than with general ones.

Note that, as reported in the examples of Figures 33.a, b, c in Appendix C, this appears to be a general behavior, mutual to all the C-LSTM-D models .

On the other side, Figure 25.b confirms results of the depth histogram in Figure 23: the C-LSTM-D model is able to predict the depth level properly and, even when the prediction is wrong, it is still close to the correct value.

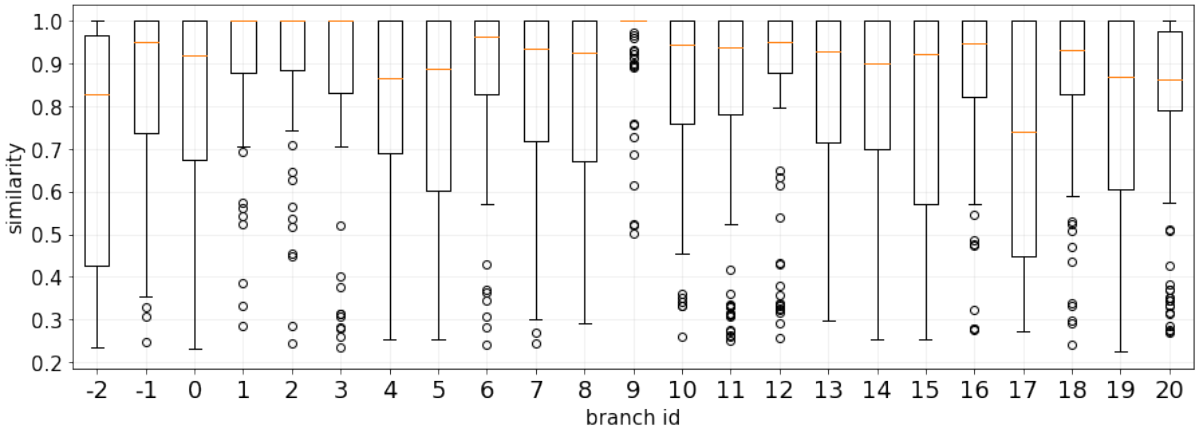


Figure 26: **Predicted similarity per branch for the testing model.**(The numerical id between 0 and 19 corresponds to the branches of Table 13 in Appendix B. The other branches are grouped together in the column corresponding to the id -1 since they were not enough populated to be represented in the boxplot (i.e. they had less than 100 entries). Similarly, id -2 corresponds to multiple branches that appear in the ontology with less than 100 elements.)

Figure 26 contains boxplots of the similarity between predicted and correct HPO classes stratified per branch. Since categories are not correlated as the levels of depth, it is more difficult to recognize trends among branches. Certainly, it appear evident that some branches, as the number 9 (*Abnormality of the skeletal system, Abnormality of limbs*) or the number 2 (*Abnormality of head or neck, Abnormality of the skeletal system*) have very good results, while other branches, as the number 8 (*Abnormality of the integument*), do not. It is interesting to note that, the behaviors of the branches transcend the single combination of parameters and is characteristic of all the model (some example in Figures 34.a,b,c in Appendix C).

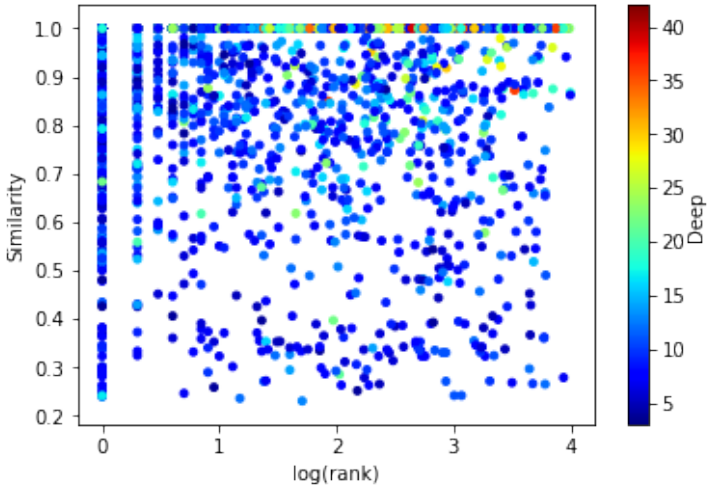


Figure 27: **Predicted similarity vs corresponding rank in the sorted list of closer vectors in** $WE_{g1\,A}$

Finally, in the scatter plot in Figure 27 there is represented the predicted similarity of some layman terms as opposed to the rank of the correct term in the sorted list of most similar vectors in $WE_{g1\,A}$ space. Since the model works better with specific terms and the WE does not, points corresponding to more specific HPO classes are concentrated in the upper-right corner. Apart from this, there is no correlation between the classes or the layman terms that the embedding can not represent appropriately (and hence have a high rank) and the layman terms that the model can not translate.

6.3.2 LSTM-D and LSTM-P models results

As mentioned before, for reasons of time and available resources, for the other two models, it was not possible to carry out an analysis as deep as the one done for the C-LSTM-D model. However, a consistent number of combinations of parameters have been tested anyway, and outcomes are coherent with the precedent ones. Results of the LSTM-D model are reported in Table 19 Appendix B, while results of the LSTM-P model are reported in Table 21.

In figure 28, a boxplot representing the performances of the LSTM-D model is reported. Each parameter combination is identified by a numerical id reported in Table 18, Appendix C. Numerical results are reported in Table 19, Appendix B. LSTM-D model in general performs better than the C-LSTM-D model. Just considering the WE_{rand} based models this trend is confirmed:

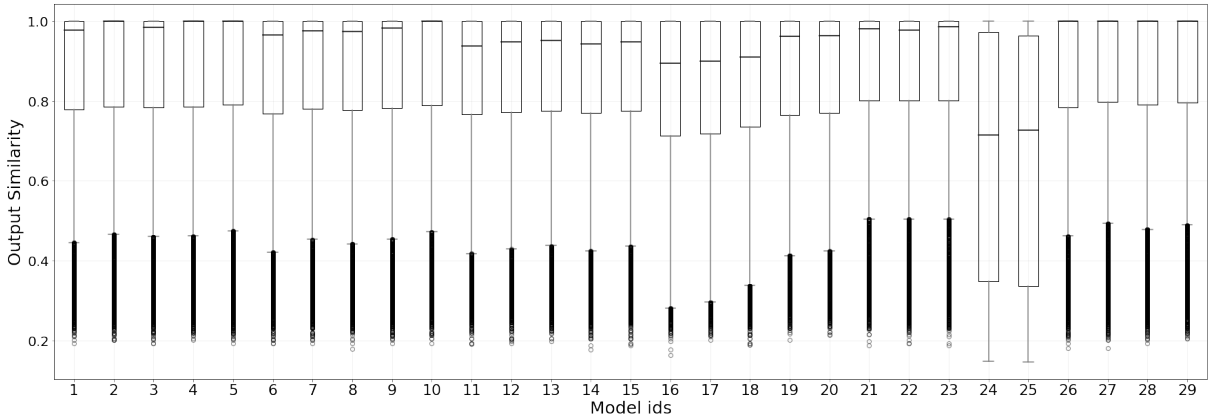


Figure 28: **Predicted similarity per model.**

C-LSTM-D($WE_{rand\ B}$, 600, 500) for example has a median similarity of 0.587, while the corresponding LSTM-D($WE_{rand\ B}$, 600, 500) has a median of 0.727. Considering better performing combinations this difference is maintained: C-LSTM-D($WE_{svd\ A}$, 600, 600) correctly classifies 48.53% of the terms, while LSTM-D($WE_{svd\ A}$, 600, 600) correctly classifies 51.45% of the entries. Statistical tests support this hypothesis (see Table 4).

Models compared	p value (Mann–Whitney U, $\alpha = 0.05$)	p value (Kruskal-Wallis, $\alpha = 0.05$)
C-LSTM-D($WE_{svd\ A}$, 600, 600) vs LSTM-D($WE_{svd\ A}$, 600, 600)	9.21e-0.5	1.5e-04

Table 4: **Results of Mann-Whitney U test and Kruskal-Wallis test for different models.**

Also the percentage of branches and depth levels correctly classified confirm that the LSTM-D model is better: looking at Figure 37.a in Appendix C, one can note that the values are similar to the ones in Figure 23 but slightly higher. In particular, the WE_{rand} based models performs over 60% for branches classification.

All the consideration exposed in Section 7.3.1 remain valid for LSTM-D models. First of all, the HPO embedding spaces behave similarly. For example WE_{g1} is the best Gen. WEs, and vectors in $\mathbb{R}^{200}$ perform better, except for the DS embedding. Furthermore, the different combinations of parameters affect the outcomes in the same way, supporting all the precedent hypotheses about the influence of the parameters on the outcomes.

Figure 29 contains two histograms indicating the percentage of correctly classified terms per depth for two models (LSTM-D($WE_{g1\ A}$, 400, 600) in green and C-LSTM-D($WE_{g1\ A}$, 400, 600) in red). The models behave similarly: LSTM-D performs slightly better in the upper levels (as seen before this model performs better in general), but the trend is identical. At lower levels the percentage of correctly classified layman terms is lower, improving linearly as the terms become more specific. As reported in some examples in Figure 35 in Appendix C, this behavior is characteristic of all the combination of parameters of the LSTM-D model. Furthermore, the

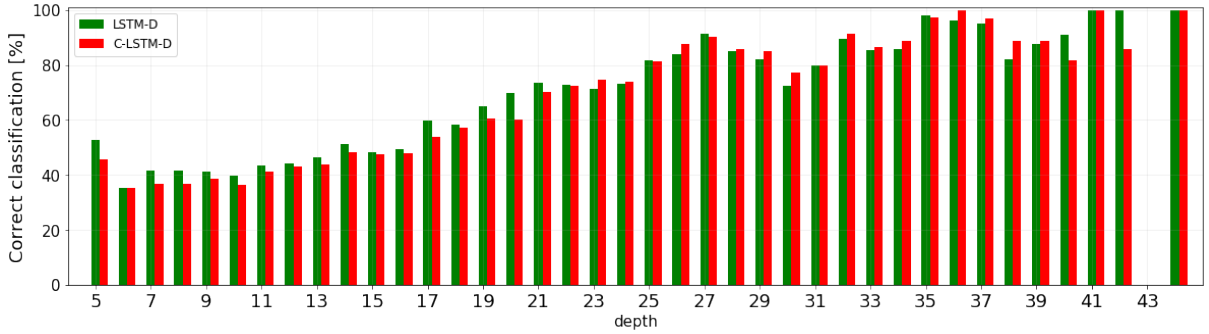


Figure 29: **Comparison between two models of the percentage of correct classified terms per depth .**

boxplots of the similarity per branches (Fig. 35.b,c) are very similar to C-LSTM-D ones (Figure 26), confirming that the models behave similarly.

To explain that the C-LSTM-D model and the LSTM-D model have very similar results,I would refer to the structure of the convolutional model. The scope of the convolutional layer is to extract features from the embedding matrix that represent the input sentence. As a consequence, the LSTM layer is fed with smaller vectors with respect to the ones that enter in the LSTM layer of the LSTM-D model. For this reason, from one side the training time is reduced, but on the other side, apparently some information contained in the embedding matrix is lost during the features extraction.

For what deal the errors across different models, it must be noted that 45% of that terms misclassified with a similarity between predicted and correct term less then 0.5 in all the C-LSTM-D models, were misclassified with the same error on prediction also in all the LSTM-D models

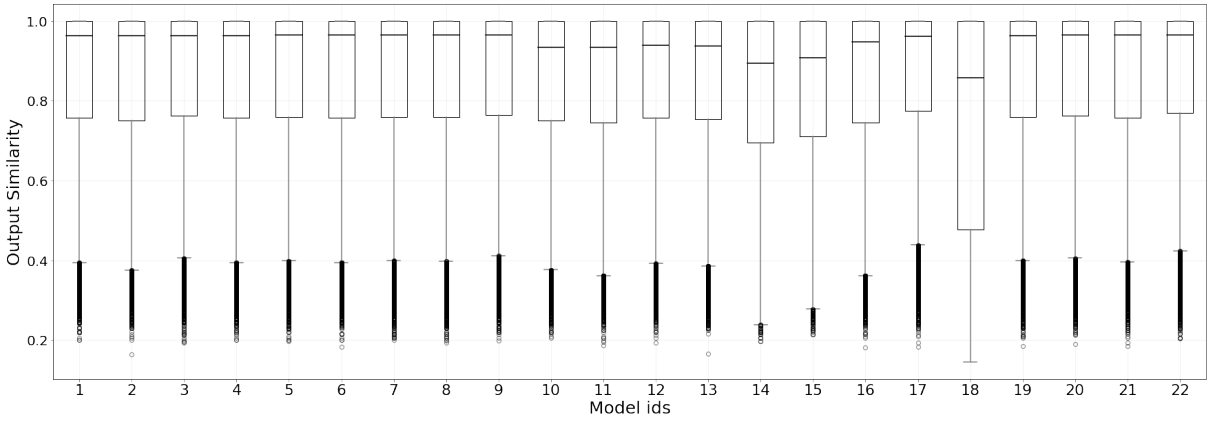


Figure 30: **Predicted similarity per model (LSTM-P).**

LSTM-P required more training time, and hence it was less investigated. Results in terms of predicted similarity for different parameter combinations are reported in Figure 30, while the

complete numerical results are reported in Table 21, Appendix B and histograms over branch and depth classification are reported in Figure 37.b, Appendix C. Analysis of the behavior of single models is shown in Figure 36. As it could be expected, LSTM-P behaves as the C-LSTM-D model. However, numerical results show no improvements in performances. The introduction of parallel dense layers has degraded the performances as compared to the simple LSTM-D model, obtaining similar results as C-LSTM-D, but with a model that is 5 times slower. A possible explanation is that not all the branches are equally represented in the training set: considering that each dense block in the parallel layer is trained with the terms of the corresponding branch alone, it could be that some of these blocks are not enough trained to reach good results. If this hypothesis is true, a richer training set representing all the categories will improve the performance of this model, probably surpassing LSTM-D .

7 Costs and environmental impact

Costs required to carry out this project can be divided in 3 categories:

- **Personnel costs**: these costs refers to the main workforce costs. In particular, it must be considered: the full time work of a graduated engineer during the whole duration of the project (4.5 months). This cost is estimated considering the remuneration of a Group 2 UPC researcher [4]; the costs of the two supervisor, estimated considering the possible pays for specialized advisors, is of 90 €/h. Personnel cost are reported in Table 5.
- **Materials**: The materials used during this project included a personal laptop (Asus tp 300L, 499.99€) plus a computer from the University for the calculations (Motherboard: ASUS PRIME B250-PLUS 249.99 €; GPU: NVIDIA TITAN Xp, 1,999€ and EVGA GeForce GTX 1070, 559€). Since the hardware was used only for a small period of time compared to their total lifespan (which here it is assumed to be 3 years), the costs were amortized for the period of usage. Material costs are reported in Table 6.
- **Energy costs**: the energy used was estimated to be 0.2 kWh for the laptop (including the desktop) and 0.8 kWh for the other computer. Furthermore, it must be considered the energy of the office light (mean consumption of 16 neon bulbs 0.48 kWh). Energy costs are reported in Table 7.

All these costs are summarized in Tables 8. The total budget needed to complete the project is estimated to be 14,965 €. As expected, since there was no need of particular hardware, most of the costs comes from salaries and personnel costs.

Employee	**Gross salary** [€/month]	**Working time** [h/months]	**Cost** [€/month]
Main engineer	1,935.90	*full time*	1,935.90
Supervisor 1	90 [€/h]	10	900
Supervisor 2	90 [€/h]	4	360
Total			3,195.9

Table 5: **Table of Personel costs**

Item	**Purchase price** [€]	**Life time** [years]	**Cost** [€/month]
Laptop	499.99	3	13.89
Motherboard	249.99	3	6.94
GPU1	1,999	3	55.53
GPU2	559	3	15.53
Total			91.89

Table 6: **Table of material costs**

[4] Data from www.upc.edu/ca/la-upc/la-institucio/fets-i-xifres/pressupost_upc_2018_final-1.pdf

Item	Energy [kWh]	Usage [h/day]	Cost [€/month]
Personal laptop	0.2	8	3.7
Computer for calculations	0.8	16	29.57
Office light	0.48	4	4.43
Total			37.7

Table 7: **Table of energy costs** (Energy cost 0.11 €/kWh, 21 days/month)

Item	Monthly costs [€/month]	Cost per projects [€]
Personnel costs	3,195.9	14,381.55
Material costs	91.89	413.5
Energy costs	37.7	169.95
Total		14,965

Table 8: **Table of total costs**

Besides the costs and the usefulness of the results of the project, it is important to evaluate the impact that this work could have on the environment. Since all the work was done using only a laptop and an additional computer, the environmental impact that the project could have comes only from the energy consumption. According to recent analysis of WWF, only 50 % of the energy produced in Spain comes from renewable resources, while the other 50 % has an impact in terms of CO2 and other pollutants emissions, since it is produced with nuclear sources or with coal [70]. As a consequence, one can estimate environmental impact as summarized in Table 9. For the calculations, it was considered that the total amount of energy used for the project was of 16.3 kWh/day, hence 1540.35 kWh for the whole project.

Type of emission	Amount per kWh	Total
Carbon dioxide (CO2)	0.06 kg/kWh	92.42 kg
Sulfure dioxide (SO2)	0.107 g/kWh	164,82 g
Nitrogen oxide (NOx)	0.083 g/kWh	127.85 g
Intermediate and low level waste	0,00259 cm3 /kWh	3.99 cm3
High level waste	0,317 mg/kWh	488.29 mg

Table 9: **Environmental impact estimation**

Conclusions

In this work novel methods for predicting specific phenotypes from generic text have been described. The main objective was to investigate the most used Natural Language Processing techniques in order to develop a machine translation tool for an automatic translation between layman and HPO terms. All the proposed methods relied on the same idea: firstly, a vector space representing HPO terms was created using different word embedding techniques; then, deep learning architectures as convolutional neural networks and LSTM recurrent neural network were used to encode layman terms in that space. The closest vector in this space was finally chosen as synonym of the input term.

Five different embedding techniques were tested: the first three, named General embeddings, were based on Word2Vec models using different techniques to combine its vectors; another approach was based on Latent Semantic Analysis and was named Domain Specific Embedding; the last techniques, the Combined embedding, was created merging a Generic Embedding with the Domain Specific one.

Besides the embedding techniques, three models have been analyzed thorougly. LSTM-D was based on a LSTM recurrent neural network and a fully-connected layer to encode layman terms in the embedding space. The second model (C-LSTM-D) introduced a convolutional layer to extract salient features and reduce training times. Finally, the last model (LSTM-P) tries to exploit the tendency of the proposed embedding space to cluster HPO categories using different fully-connected layers in parallel.

The different types of embeddings and different models have been tested with cross validation, using performance metrics defined specifically for the HPO domain, such as categories or similarity among classes. Even if results present differences among models and embeddings, most of the models had satisfactory performances. Best results were achieved with the LSTM-D model with the Combined embedding, being able to exactly identify the correct phenotype in validation in more than half of the cases, with a mean similarity between predicted and correct HPO term close to 0.9.

In the future the model can be improved exploring new configurations as well as input and output spaces. Furthermore, only a limited part of the tunable parameters has been taken into consideration: a more detailed work on the influence of each parameter on the outcomes should be of central attention to improve model performance. Eventually, training and testing were carried out only with terms and sentences inside HPO. This is a limiting aspect: firstly from the point of view of the amount of data; and secondly, for the fact that synonyms inside HPO could not represent laymen vocabulary effectively. Future efforts should be focused on extending the available training set with terms from other sources (Wikipedia, patients' forum, social networks, etc.).

Bibliografia

[1] P. N. Robinson, "Deep phenotyping for precision medicine," *Hum Mutat.*, vol. 33, no. 5, pp. 770–780, 2012.

[2] S. Baldovino, A. Moliner, D. Taruscio, E. Daina, and D. Roccatello, "Rare diseases in europe: From a wide to a local perspective," *Israel Medical Association Journal*, vol. 18, pp. 359–363, 6 2016.

[3] Eurordis, "Survey of the delay in diagnosis for 8 rare diseases in europe," tech. rep., 2007.

[4] N. F. Noy and D. L. McGuinness, "Ontology development 101 : A guide to creating your first ontology," 2001.

[5] G. A. Miller, "Wordnet: A lexical database for english," *Commun. ACM*, vol. 38, pp. 39–41, Nov. 1995.

[6] M. Stearns, C. Price, K. Spackman, and A. Wang, "Snomed clinical terms: Overview of the development process and project status," *Proceedings / AMIA ... Annual Symposium. AMIA Symposium*, pp. 662–6, 02 2001.

[7] G. L. G. Consortium, J. Richter, B. Drosophila, G. Project, and G. M. Rubin, "Creating the gene ontology resource: design and implementation.," *Genome research*, vol. 11 8, pp. 1425–33, 2001.

[8] J. Jia, R. Wang, Z. An, Y. Guo, X. Ni, and T. Shi, "Rdad: A machine learning system to support phenotype-based rare disease diagnosis," *Front Genet*, vol. 9, p. 587, 2018.

[9] S. Köhler, S. C. Doelken, C. J. Mungall, S. Bauer, and H. V. F. et al., "The human phenotype ontology project: linking molecular biology and disease through phenotype data," *Nucleic Acids Research*, vol. 42, pp. D966–974, 2014.

[10] N. Seco, T. Veale, and J. Hayes, "An intrinsic information content metric for semantic similarity in wordnet," in *Proceedings of the 16th European Conference on Artificial Intelligence*, ECAI'04, pp. 1089–1090, 2004.

[11] J. Jiang and D. Conrath, "Semantic similarity based on corpus statistics and lexical taxonomy," in *Proceedings of the International Conference on Research in Computational Linguistics*, 1998.

[12] P. Resnik, "Using information content to evaluate semantic similarity in a taxonomy," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, (San Francisco, CA, USA), pp. 448–453, Morgan Kaufmann Publishers Inc., 1995.

[13] S. Joseph, K. Sedimo, F. Kaniwa, H. Hlomani, and K. Letsholo, "Natural language processing: A review," *Natural Language Processing: A Review*, vol. 6, pp. 207–210, 03 2016.

[14] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing [review article]," *IEEE Computational Intelligence Magazine*, vol. 13, p. 55–75, Aug 2018.

[15] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[16] P. Koehn, *Statistical Machine Translation*. Cambridge University Press, 2009.

[17] Y. Wang, S. Liu, N. Afzal, M. Rastegar-Mojarad, L. Wang, F. Shen, P. Kingsbury, and H. Liu,

"A comparison of word embeddings for the biomedical natural language processing," *Journal of Biomedical Informatics*, vol. 87, p. 12–20, Nov 2018.

[18] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, 1990.

[19] G. Salton and C. Buckley, "Term weighting approaches in automatic text retrieval," *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, 1988.

[20] E. Altszyler, S. Ribeiro, M. Sigman, and D. Fernández Slezak, "The interpretation of dream meaning: Resolving ambiguity using latent semantic analysis in a small corpus of text," *Consciousness and Cognition*, vol. 56, p. 178–187, Nov 2017.

[21] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of EMNLP*, p. 1532–1543, 2014.

[22] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, Nov. 2011.

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[24] M. Sahlgren, "The distributional hypothesis," 2008.

[25] X. Rong, "word2vec parameter learning explained," 2014.

[26] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," 2013.

[27] W. Yin and H. Schütze, "Learning word meta-embeddings," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (Berlin, Germany), pp. 1351–1360, Association for Computational Linguistics, Aug. 2016.

[28] P. Kameswara Sarma, Y. Liang, and B. Sethares, "Domain adapted word embeddings for improved sentiment classification," in *Proceedings of the Workshop on Deep Learning Approaches for Low-Resource NLP*, (Melbourne), pp. 51–59, Association for Computational Linguistics, July 2018.

[29] M. Pilehvar and N. Collier, "Improved semantic representation for domain-specific entities," in *Proceedings of the 4th BioNLP Shared Task Workshop*, pp. 12–16, 2016.

[30] R. Navigli and S. P. Ponzetto, "Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network," *Artif. Intell.*, vol. 193, pp. 217–250, 2012.

[31] W. J. Hutchins, "Machine translation: A brief history," 1995.

[32] L. Deng and D. Yu, "Deep learning: Methods and applications," Tech. Rep. MSR-TR-2014-21, May 2014.

[33] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1746–1751, Association for Computational Linguistics, Oct. 2014.

[34] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," in *Proceedings of the 52nd Annual Meeting of the Association for Compu-*

tational Linguistics (Volume 1: Long Papers), (Baltimore, Maryland), pp. 655–665, Association for Computational Linguistics, June 2014.

[35] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," 2015.

[36] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, 2008.

[37] B. Hu, Z. Tu, Z. Lu, H. Li, and Q. Chen, "Context-dependent translation selection using convolutional neural network," *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, 2015.

[38] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Artif. Intell.*, vol. 521, pp. 236–444, 2015.

[39] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 1997.

[40] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, Oct. 2014.

[41] P. Koehn, "Neural machine translation," 2017.

[42] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 3104–3112, Curran Associates, Inc., 2014.

[43] O. Vinyals and Q. Le, "A neural conversational model," 2015.

[44] C. Zhou, C. Sun, Z. Liu, and F. C.-M. Lau, "A c-lstm neural network for text classification," *CoRR*, vol. abs/1511.08630, 2015.

[45] V. V. Vydiswaran, Q. Mei, D. A. Hanauer, and K. Zheng, "Mining consumer health vocabulary from community-generated text," in *AMIA Annual Symposium proceedings. AMIA Symposium vol.*, 2016.

[46] M. Baroni and S. Siri, "Using cooccurrence statistics and the web to discover synonyms in a technical language," in *Proceedings of the Fourth International Conference on Language Resources and Evaluation*, (LREC'04), 2004.

[47] M. Hagiwara, Y. Ogawa, and K. Toyama, "Selection of effective contextual information for automatic synonym acquisition," in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, (ACL-44), pp. 353–360, 2006.

[48] G. Soğancıoğlu, H. Özturk, and A. Özgü, "Biosses: a semantic sentence similarity estimation system for the biomedical domain," *Bioinformatics*, vol. 33, no. 14, pp. i49–i58, 2017.

[49] E. Ukkonen, "Approximate string-matching with q-grams and maximal matches," *Theoretical Computer Science*, vol. 92, pp. 191–211, 01 1992.

[50] L. R. Lawlor, "Overlap, similarity, and competition coefficients," *Ecology*, vol. 61, p. 245, 04 1980.

[51] O. Bodenreider, "The unified medical language system (umls): integrating biomedical terminology," *Nucleic acids research*, vol. 32 Database issue, pp. D267–70, 2004.

[52] Q. Chen, Y. Peng, and Z. Lu, "Biosentvec: creating sentence embeddings for biomedical texts," 2018.

[53] M. Pagliardini, P. Gupta, and M. Jaggi, "Unsupervised learning of sentence embeddings using compositional n-gram features," *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018.

[54] N. A. Vasilevsky, M. E. Engelstad, E. D. Foster, M. A. Haendel, P. Robinson, S. Köhler, and C. J. Mungall, "Enhancing the human phenotype ontology for use by the layperson," in *International Conference on Biological Ontology & BioCreative*, 2016.

[55] S. Köhler, L. Carmody, and N. V. et al., "Expansion of the human phenotype ontology (hpo) knowledge base and resources," *Nucleic Acids Research*, vol. 8, no. 47, pp. D1018–D1027, 2018.

[56] N. Vasilevsky, E. Foster, and M. E. et al., "Plain-language medical vocabulary for precision diagnosis," *Nature Genetics*, vol. 50, pp. 474–476, 2018.

[57] F. Dhombres and O. Bodenreide, "Interoperability between phenotypes in research and healthcare terminologies—investigating partial mappings between hpo and snomed ct," *Journal of Biomedical Semantics*, vol. 7, no. 3, 2016.

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

[59] R. McDonald, G. Brokos, and I. Androutsopoulos, "Deep relevance ranking using enhanced document-query interactions," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2018),*, 2018.

[60] P. Lafon, "Sur la variabilité de la fréquence des formes dans un corpus," *Mots*, vol. 1, pp. 127–165, 01 1980.

[61] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st ed., 2009.

[62] F. Chollet, "keras." `https://github.com/fchollet/keras`, 2015.

[63] A. C. Heusser, K. Ziman, L. L. W. Owen, and J. R. Manning, "Hypertools: a python toolbox for gaining geometric insights into high-dimensional data," *Journal of Machine Learning Research*, vol. 18, no. 152, pp. 1–6, 2018.

[64] I. Jolliffe, *Principal Component Analysis*. Springer Verlag, 1986.

[65] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[66] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theor.*, vol. 13, pp. 21–27, Sept. 2006.

[67] J. Bradbury, S. Merity, C. Xiong, and R. Socher, "Quasi-Recurrent Neural Networks," *arXiv e-prints*, p. arXiv:1611.01576, Nov 2016.

[68] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Annals of Mathematical Statistics*, vol. 18, 11 1946.

[69] W. H. Kruskal and W. Allen Wallis, "Use of ranks in one-criterion variance analysis," *Journal of The American Statistical Association*, vol. 47, pp. 583–621, 12 1952.

[70] R. G. Monzon, "Observatorio de la electricidad mayo 2016.," tech. rep., WWF, 2016.

Supplementary material

Appendix A: Algorithms

List of algorithms used.

Input: text t, boolean *lsa*, list *'coupled words'*
text = lower(text);
for *word* $w \in t$ **do**
  if *is ordinal number(w)* **then**
    w = ordinal to string(w);
  end
  if *is cardinal number(w)* **then**
    w = cardinal to string(w);
  end
  if $w =$ "$d[d] * [-]d[d] *$ " **then**
    w1,w1 = separe w on '-';
    w = ordinal to string(w1)+' '+ordinal to string(w2);
  end
  if *punctuation* ($\neq$ '-') $\in w$ **then**
    w = clean punctuation(w);
  end
  if *lsa* **then**
    if $w \in$ *stop words* **then**
      eliminate w;
    end
    if '-'$\in w$ **then**
      if $w \notin$ *coupled word* **then**
        separe w on '-';
      end
      else
        eliminate '-' ;
      end
    end
  end
end

Algorithm 1: Pre-process a text t
(Boolean input *lsa* indicate if the algorithm is used for pre-processing an input of a model or a text for the WE_{LSA})

Input: HPO term t_i, *'coupled words'* list
$R_t(t_i)$ = empty list;
for $w \in t_i$ **do**
    w = lower(w);
    if '-'$\in w$ **then**
        if $w \notin coupled\ word$ **then**
            separe w on '-';
        end
    end
    if $w \in$ *stop words* **then**
        eliminate w;
    end
    if *punctuation* ($\neq$ '-') $\in w$ **then**
        w = clean punctuation(w);
    end
    $R_t \leftarrow w$;
end

Algorithm 2: Extract R_t list

Input: HPO term t_i, $R_t(t_i)$, n, l
pages = empty list;
Wiki = Wikipedia corpus;
for $w \in R_t(i)$ **do**
    $pg_{current} \leftarrow$ Wiki first n pages related to w;
    pages $\leftarrow pg_{current}$;
    for $pp \in pg_{current}$ **do**
        pages $\leftarrow$ first l links in pp;
    end
end
$D_t(t_i)$ = concatenate (pages)

Algorithm 3: Extract Wikipedia related pages

Input: HPO term t_i, $D_t(t_i)$, medical dictionary MD
R_t^* = empty list;
for $w \in D_t(t_i)$ **do**
  if $w \in MD$ **then**
    F = frequency of w in Wiki;
    f = frequency of w in $D_t(t_i)$;
    T = # of words in Wiki;
    t = # of words in $D_t(t_i)$.;
    X = hypergeometric dist.(T,t,F);
    spec$_w$ = $logP(X \geq f)$;
    if $spec_w > th$ **then**
      $R_t^* \leftarrow w$ (according to spec$_w$)
    end
  end
end

Algorithm 4: Extract HPO related words

Appendix B: Tables

Original term	Alternative
hyperasparaginemia	high blood asparagine level
acromelia	shortening acromelic
hypocystinemia	low blood cystine level
hyperthreoninuria	high urine threonine level
hyperisoleucinemia	high blood isoleucine level
acromelia	shortening acromelic
aphalangy	aphalangia
hyposerinemia	low blood serine level
hyperbetaalaninemia	high blood beta-alanine level
diastomatomyelia	longitudinal split spinal cord
polyminimyoclonus	small amplitude myoclonus
aphalangy	aphalangia
tympanii	tympani
plalelet	platelet
amyelia	absent spinal cord
hypoleucinemia	low blood leucine level
quelprud	post auricular cartilaginous
afebril	afebrile
diaminoaciduria	diamino aciduria
bilambdoid	vilateral lambdoid
tympanii	tympani
bronchomegaly	abnormal bronchi widening
phosphohydroxylysinuria	high urine phosphohydroxylysine level
tracheobronchmegaly	abnormal trachea bronchi
hypomanganesemia	low blood manganese level
nonopposable	non opposable
undermodelled	under modelled
incyclophoria	inward cyclophoria
atheroeruptive	athero eruptive
hypoglutamatemia	low blood glutamate level
hypercystinemia	high blood cystine level
hypoplastia	hipoplasia
hydroxyphenylpyruvic	hydroxyphenylpyruvate
microthorax	micro thorax
lacticaciduria	high urine lactic acid level
platonychia	abnormal flat nail
underfolded	under folded
polycalycosis	poly calices

Table 10: **Words not represented in McDonalds' WE and proposed alternatives** (I)

Original term	Alternative
triplomyelia	triplication spinal cord
keratocytosis	keratocytes
mediosternal	medio sternal
circumungual	around nails
hyperchloriduria	increased urinary chloride
dysarthic	dysarthria
underfolded	under folded
neryous	nervous
undermodelled	under modelled
acrobrachycephaly	acro brachycephaly
methylglutaric	methylglutaconic
spillus	spilus
overtubulated	over tubulated
kienboeck	lunatomalacia
hypetropia	hypertropia
hyperglutaminuria	high glutamine level
hypochloriduria	low urine chloride level
brachytelomesophalangy	brachymesophalangy
hypoammonemia	low blood ammonia level
supraauricular	supra auricular
isothenuria	isosthenuria
serpentinum	serpiginosum
hypothreoninemia	low blood threonine level
hyperhistidinemia	high blood histidine level

Table 11: **Words not represented in McDonalds' WE and proposed alternatives** (II)

Parameter	LSTM-D model	C-LSTM-D model	LSTM-P model
HPO WE space	**all WE**	**all WE**	**all WE**
HPO WE dim.	**200/400**	**200/400**	**200/400**
Embedding layer dim.	**400/600**	**400/600**	**400/600**
LSTM dim.	**400/500/600**	**400/600**	**400/600**
Dense layer activation func.	linear	linear	linear+sigmoid
Number of conv. filters	0	600	400
Filters size	-	5	5
Conv. activation func.	-	relu	relu
Max pooling	-	15	15
Dropout	-	0	-
Batch size	64	64	64
Training epochs	25	25	25

Table 12: **Parameters per model. Bold rows indicate tunable parameters.** (Parameters not in this table are the Keras default ones).

id	branch
0	*Abnormality of the genitourinary system*
1	*Abnormality of head or neck*
2	*Abnormality of head or neck, Abnormality of the skeletal system*
3	*Abnormality of the ear*
4	*Abnormality of the eye*
5	*Abnormality of the nervous system*
6	*Abnormality of the endocrine system*
7	*Abnormality of the skeletal system*
8	*Abnormality of the integument*
9	*Abnormality of the skeletal system, Abnormality of limbs*
10	*Abnormality of the musculature*
11	*Abnormality of the digestive system*
12	*Growth abnormality*
13	*Abnormality of the cardiovascular system*
14	*Abnormality of blood and blood-forming tissues*
15	*Abnormality of metabolism/homeostasis*
16	*Abnormality of head or neck, Abnormality of the integument*
17	*Abnormality of the respiratory system*
18	*Abnormality of blood and blood-forming tissues, Abnormality of the immune system*
19	*Abnormality of the genitourinary system, Abnormality of metabolism/homeostasis*
20	*Abnormality of the immune system*
21	*Clinical course*
22	*Neoplasm*
23	*Abnormality of limbs*
24	*Abnormality of connective tissue*
25	*Mode of inheritance*
26	*Abnormal cellular phenotype*
27	*Constitutional symptom*
28	*Abnormality of the breast*
29	*Clinical modifier*
30	*Abnormality of the voice*
31	*Abnormality of prenatal development or birth*
32	*Abnormality of the thoracic cavity*
33	*Frequency*

Table 13: **List of branches and corresponding identification numbers.**

WE	A	B
g1	0.794 ± 0.178	0.801 ± 0.178
g2	$\mathbf{0.814 \pm 0.166}$	0.817 ± 0.169
g3	0.766 ± 0.200	0.772 ± 0.197
lsa	0.801 ± 0.180	$\mathbf{0.820 \pm 0.167}$
svd	0.797 ± 0.174	0.804 ± 0.174
rand	0.355 ± 0.104	0.352 ± 0.099

Table 14: **Mean similarity ($\pm$ std) of the boxes in figure 21**

Mod.	Min. param (LSTM = 400, inp WE=400)	Max. param. (LSTM = 600, inp WE=600)	Time to train [min]
LSTM-D	6.7902×10^4	11.2654×10^4	[30-40]
C-LSTM-D	8.3108×10^4	13.066×10^4	[5-10]
LSTM-P	21.223126×10^4	23.828326×10^4	[35-50]

Table 15: **Number of parameters per model**

Id	Model (WE_j,WE inp.,LSTM)
1	g1A 400 400
2	g1A 400 500
3	g1A 400 600
4	g1A 600 400
5	g1A 600 500
6	g1A 600 600
7	g1B 400 400
8	g1B 400 500
9	g1B 400 600
10	g1B 600 400
11	g1B 600 500
12	g1B 600 600
13	g2A 400 400
14	g2A 400 600
15	g2A 600 400
16	g2A 600 500
17	g2A 600 600
18	g2B 400 400
19	g2B 400 600
20	g2B 600 400
21	g2B 600 600
22	g3A 400 400
23	g3A 400 600
24	g3A 600 400
25	g3A 600 500
26	g3A 600 600
27	g3B 600 400
28	g3B 600 500
29	g3B 600 600

(a)

Id	Model (WE_j,WE inp.,LSTM)
30	lsaA 400 400
31	lsaA 400 600
32	lsaA 600 400
33	lsaA 600 500
34	lsaA 600 600
35	lsaB 400 400
36	lsaB 400 600
37	lsaB 600 400
38	lsaB 600 500
39	lsaB 600 600
40	randA 400 400
41	randA 400 600
42	randA 600 400
43	randA 600 600
44	randB 400 400
45	randB 400 600
46	randB 600 400
47	randB 600 500
48	randB 600 600
49	svdA 400 400
50	svdA 400 600
51	svdA 600 400
52	svdA 600 600
53	svdB 400 400
54	svdB 400 600
55	svdB 600 400
56	svdB 600 500
57	svdB 600 600

(b)

Table 16: **List of C-LSTM-D models and corresponding id numbers.**

Model (WE_j,WE inp.,LSTM)	Corr.[%]	Near[%]	Wrong[%]	Median sim.	Mean sim. ($\pm$ std)	Corr. br.[%]	Corr. deep[%]
g1A 400 400	46.71	80.21	19.79	0.963	0.851 $\pm$ 0.212	90.42	76.28
g1A 400 500	47.26	80.67	19.33	0.965	0.853 $\pm$ 0.212	90.42	76.64
g1A 400 600	**47.88**	80.8	19.2	**0.967**	**0.854 $\pm$ 0.211**	90.65	**77.04**
g1A 600 400	47.34	80.52	19.48	0.965	**0.854 $\pm$ 0.211**	90.51	76.42
g1A 600 500	47.54	80.73	19.27	0.966	**0.854 $\pm$ 0.211**	90.56	76.52
g1A 600 600	47.74	80.55	19.45	0.966	0.853 $\pm$ 0.211	90.49	76.8
g1B 400 400	45.03	78.51	21.49	0.953	0.839 $\pm$ 0.222	89.06	74.24
g1B 400 500	45.61	78.97	21.03	0.957	0.843 $\pm$ 0.219	89.68	74.8
g1B 400 600	47.77	80.69	19.31	0.966	**0.854 $\pm$ 0.211**	90.63	76.88
g1B 600 400	45.48	79.13	20.87	0.957	0.843 $\pm$ 0.218	89.73	75.2
g1B 600 500	45.7	79.04	20.96	0.957	0.843 $\pm$ 0.22	89.39	74.96
g1B 600 600	47.13	79.69	20.31	0.964	0.848 $\pm$ 0.218	89.82	75.74
g2A 400 400	39.69	80.33	19.67	0.931	0.845 $\pm$ 0.206	90.36	73.04
g2A 400 600	40.64	80.63	19.37	0.936	0.847 $\pm$ 0.205	90.65	73.64
g2A 600 400	40.84	80.92	19.08	0.937	0.848 $\pm$ 0.205	90.64	73.64
g2A 600 500	41.12	80.85	19.15	0.937	0.848 $\pm$ 0.205	90.56	73.63
g2A 600 600	41.08	**81.21**	18.79	0.938	0.85 $\pm$ 0.203	**90.98**	73.46
g2B 400 400	38.16	78.98	21.02	0.927	0.836 $\pm$ 0.213	89.5	72.12
g2B 400 600	39.6	79.98	20.02	0.932	0.843 $\pm$ 0.208	90.36	72.67
g2B 600 400	38.4	79.45	20.55	0.929	0.838 $\pm$ 0.211	89.74	71.75
g2B 600 600	39.92	80.21	19.79	0.933	0.843 $\pm$ 0.209	90.11	72.93
g3A 400 400	32.73	76.26	23.74	0.893	0.816 $\pm$ 0.215	88.76	70.4
g3A 400 600	34.6	76.94	23.06	0.902	0.823 $\pm$ 0.213	89.64	71.25
g3A 600 400	32.98	76.8	23.2	0.897	0.819 $\pm$ 0.214	88.97	70.45
g3A 600 500	34.06	76.56	23.44	0.901	0.82 $\pm$ 0.215	89.15	71.05
g3A 600 600	35.33	77.01	22.99	0.904	0.822 $\pm$ 0.215	89.38	71.73
g3B 600 400	33.29	76.68	23.32	0.898	0.819 $\pm$ 0.213	89.31	70.69
g3B 600 500	34.15	77.12	22.88	0.902	0.821 $\pm$ 0.214	89.31	71.42
g3B 600 600	35.26	77.24	22.76	0.907	0.824 $\pm$ 0.214	89.48	71.57
lsaA 400 400	44.77	80.31	19.69	0.951	0.846 $\pm$ 0.209	90.26	76.84
lsaA 400 600	44.43	80.31	19.69	0.949	0.846 $\pm$ 0.209	90.36	77.09
lsaA 600 400	44.75	80.26	19.74	0.951	0.847 $\pm$ 0.208	90.22	76.85
lsaA 600 500	44.83	80.17	19.83	0.952	0.846 $\pm$ 0.211	90.04	76.96
lsaA 600 600	44.41	80.07	19.93	0.95	0.845 $\pm$ 0.211	90.09	76.93
lsaB 400 400	46.33	**82.37**	17.63	0.962	0.858 $\pm$ 0.203	**91.08**	78.14
lsaB 400 600	46.02	81.89	18.11	0.959	0.855 $\pm$ 0.203	90.71	78.02
lsaB 600 400	46.26	82.28	17.72	0.962	0.857 $\pm$ 0.202	90.96	78.3
lsaB 600 500	46.28	82.19	17.81	0.961	0.856 $\pm$ 0.204	90.85	78.14
lsaB 600 600	**46.88**	82.34	17.66	**0.963**	**0.858 $\pm$ 0.202**	90.7	**78.3**
randA 400 400	20.27	42.25	57.75	0.484	0.605 $\pm$ 0.303	51.99	49.5
randA 400 600	21.5	44.36	55.64	0.531	0.619 $\pm$ 0.303	53.58	50.92
randA 600 400	20.22	42.4	57.6	0.485	0.599 $\pm$ 0.31	51.44	47.14
randA 600 600	22.15	45.46	54.54	0.557	0.624 $\pm$ 0.306	55.42	49.66
randB 400 400	19.85	41.31	58.69	0.484	0.603 $\pm$ 0.3	56.11	48.5
randB 400 600	21.8	45.69	54.31	0.568	0.625 $\pm$ 0.305	55.44	49.71
randB 600 400	20.9	43.93	56.07	0.523	0.615 $\pm$ 0.303	52.28	50.65
randB 600 500	21.67	46.45	53.55	**0.587**	**0.631 $\pm$ 0.302**	**55.68**	49.68
randB 600 600	**22.5**	**46.88**	53.12	0.586	0.631 $\pm$ 0.305	55.67	**51.13**
svdA 400 400	47.53	80.83	19.17	0.965	0.854 $\pm$ 0.211	90.55	77.11
svdA 400 600	47.93	80.76	19.24	0.967	0.854 $\pm$ 0.211	90.62	76.51
svdA 600 400	47.99	80.96	19.04	0.967	0.855 $\pm$ 0.211	90.51	76.78
svdA 600 600	**48.53**	81.26	18.74	**0.971**	**0.858 $\pm$ 0.208**	**91.13**	77.12
svdB 400 400	47.17	80.6	19.4	0.965	0.853 $\pm$ 0.211	90.44	76.79
svdB 400 600	48.14	80.95	19.05	0.969	0.855 $\pm$ 0.21	90.7	**77.55**
svdB 600 400	47.37	80.96	19.04	0.965	0.855 $\pm$ 0.208	90.83	77.24
svdB 600 500	47.69	80.79	19.21	0.966	0.855 $\pm$ 0.21	90.51	77.18
svdB 600 600	48.29	**81.3**	18.7	0.97	0.857 $\pm$ 0.208	90.86	77.49

Table 17: **Results for C-LSTM-D model**.

Id	Model (WE_j,WE inp.,LSTM)
1	g1A 400 400
2	g1A 400 600
3	g1A 600 400
4	g1A 600 500
5	g1A 600 600
6	g1B 400 400
7	g1B 400 600
8	g1B 600 400
9	g1B 600 500
10	g1B 600 600
11	g2A 400 400
12	g2A 400 600
13	g2A 600 600
14	g2B 600 500
15	g2B 600 600

(a)

Id	Model (WE_j,WE inp.,LSTM)
16	g3A 400 400
17	g3A 400 600
18	g3B 600 500
19	lsaA 400 600
20	lsaA 600 600
21	lsaB 400 600
22	lsaB 600 500
23	lsaB 600 600
24	randA 600 500
25	randB 600 500
26	svdA 400 600
27	svdA 600 600
28	svdB 600 500
29	svdB 600 600

(b)

Table 18: **List of LSTM-D models and corresponding id numbers.**

Model (WE_j,WE inp.,LSTM)	Corr.[%]	Near[%]	Wrong[%]	Median sim.	Mean sim. ($\pm$ std)	Corr. br.[%]	Corr. deep[%]
g1A 400 400	49.54	80.4	19.6	0.978	0.854 ± 0.215	90.12	76.78
g1A 400 600	50.25	80.77	19.23	1.0	0.858 ± 0.213	90.13	77.26
g1A 600 400	49.81	80.72	19.28	0.983	0.856 ± 0.215	89.88	77.3
g1A 600 500	50.28	80.74	19.26	1.0	0.857 ± 0.214	90.1	77.24
g1A 600 600	50.83	81.11	18.89	1.0	0.86 ± 0.212	90.43	77.57
g1B 400 400	47.71	79.61	20.39	0.966	0.847 ± 0.22	89.23	75.53
g1B 400 600	49.34	80.4	19.6	0.976	0.854 ± 0.217	89.87	76.55
g1B 600 400	49.09	80.11	19.89	0.975	0.851 ± 0.219	89.63	76.21
g1B 600 500	49.77	80.54	19.46	0.982	0.853 ± 0.218	89.58	76.72
g1B 600 600	50.29	80.64	19.36	1.0	0.857 ± 0.216	90.14	76.98
g2A 400 400	40.8	79.85	20.15	0.938	0.842 ± 0.215	89.54	73.04
g2A 400 600	42.4	79.98	20.02	0.947	0.845 ± 0.214	89.72	73.89
g2A 600 600	43.42	80.44	19.56	0.951	0.848 ± 0.213	89.84	74.18
g2B 600 500	41.65	79.55	20.45	0.943	0.842 ± 0.217	89.19	73.37
g2B 600 600	42.73	80.01	19.99	0.949	0.845 ± 0.216	89.39	73.43
g3A 400 400	32.33	75.97	24.03	0.895	0.814 ± 0.218	88.49	70.31
g3A 400 600	33.65	76.2	23.8	0.9	0.818 ± 0.218	88.69	70.85
g3B 600 500	35.69	77.78	22.22	0.91	0.826 ± 0.214	89.41	71.91
lsaA 400 600	46.54	80.12	19.88	0.962	0.849 ± 0.212	90.02	76.59
lsaA 600 600	47.06	80.69	19.31	0.964	0.851 ± 0.211	90.28	77.23
lsaB 400 600	49.65	82.91	17.09	0.981	0.864 ± 0.205	90.92	79.29
lsaB 600 500	49.48	82.6	17.4	0.978	0.862 ± 0.208	90.45	79.1
lsaB 600 600	49.91	82.83	17.17	0.986	0.864 ± 0.205	91.01	79.53
randA 600 500	23.71	50.63	49.37	0.714	0.666 ± 0.294	64.95	53.62
randB 600 500	21.05	51.2	48.8	0.727	0.659 ± 0.298	64.35	52.3
svdA 400 600	50.27	80.7	19.3	1.0	0.857 ± 0.215	90.31	77.32
svdA 600 600	51.45	81.83	18.17	1.0	0.862 ± 0.211	90.6	77.83
svdB 600 500	50.72	81.16	18.84	1.0	0.86 ± 0.212	90.4	78.22
svdB 600 600	51.28	81.56	18.44	1.0	0.862 ± 0.211	90.67	77.76

Table 19: **Results for LSTM-D model**.

Id	Model (WE_j,WE inp.,LSTM)
1	g1A 400 400
2	g1A 400 500
3	g1A 400 600
4	g1A 600 500
5	g1A 600 600
6	g1B 400 400
7	g1B 400 600
8	g1B 600 500
9	g1B 600 600
10	g2A 400 400
11	g2A 600 600

(a)

Id	Model (WE_j,WE inp.,LSTM)
12	g2B 600 500
13	g2B 600 600
14	g3A 400 600
15	g3B 600 500
16	lsaA 400 400
17	lsaB 600 600
18	randB 600 500
19	svdA 400 600
20	svdA 600 600
21	svdB 400 600
22	svdB 600 600

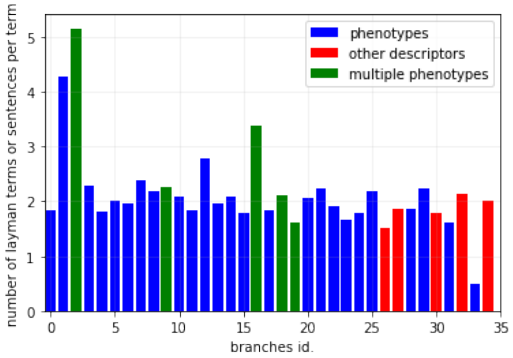
(b)

Table 20: **List of LSTM-P models and corresponding id numbers.**

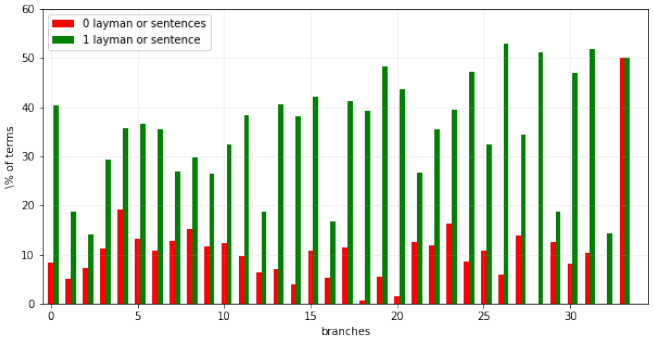
Model (WE_j,WE inp.,LSTM)	Corr.[%]	Near[%]	Wrong[%]	Median sim.	Mean sim. ($\pm$ std)	Corr. br.[%]	Corr. deep[%]
g1A 400 400	47.21	78.83	21.17	0.964	0.849 ± 0.214	90.13	76.31
g1A 400 500	46.91	78.42	21.58	0.963	0.846 ± 0.215	90.02	75.95
g1A 400 600	47.16	79.07	20.93	0.964	0.848 ± 0.214	90.18	75.99
g1A 600 500	47.18	79.07	20.93	0.963	0.848 ± 0.214	90.04	76.21
g1A 600 600	47.57	78.88	21.12	0.965	0.848 ± 0.216	90.01	76.85
g1B 400 400	47.47	78.69	21.31	0.965	0.846 ± 0.219	89.69	76.37
g1B 400 600	47.24	78.74	21.26	0.965	0.847 ± 0.218	89.7	76.4
g1B 600 500	47.51	79.01	20.99	0.965	0.847 ± 0.217	89.72	76.35
g1B 600 600	47.72	79.24	20.76	0.966	0.85 ± 0.215	90.19	76.2
g2A 400 400	40.77	78.55	21.45	0.934	0.839 ± 0.213	89.87	72.93
g2A 600 600	40.74	78.28	21.72	0.934	0.838 ± 0.214	89.57	72.69
g2B 600 500	40.97	79.16	20.84	0.939	0.841 ± 0.212	89.81	73.36
g2B 600 600	41.03	78.71	21.29	0.938	0.84 ± 0.214	89.6	72.66
g3A 400 600	33.3	74.55	25.45	0.894	0.811 ± 0.22	88.54	70.43
g3B 600 500	35.78	75.71	24.29	0.907	0.821 ± 0.217	89.09	72.14
lsaA 400 400	44.45	78.74	21.26	0.949	0.842 ± 0.211	90.02	75.53
lsaB 600 600	46.03	80.79	19.21	0.962	0.854 ± 0.206	90.71	76.98
randB 600 500	29.3	61.05	38.95	0.858	0.735 ± 0.275	76.35	60.77
svdA 400 600	47.23	79.01	20.99	0.964	0.848 ± 0.215	90.1	76.3
svdA 600 600	47.19	79.25	20.75	0.965	0.849 ± 0.214	90.05	76.35
svdB 400 600	47.4	78.95	21.05	0.965	0.848 ± 0.215	90.12	76.26
svdB 600 600	47.5	80.01	19.99	0.965	0.851 ± 0.213	90.29	76.96

Table 21: **Results for LSTM-P model**.

Appendix C: Additional images



(a) *Number of entries in the training set per term per branch*



(b) *% of term with 0 or 1 entries in the training set.*

Figure 31: **Distribution of entries in the training set inside HPO.**

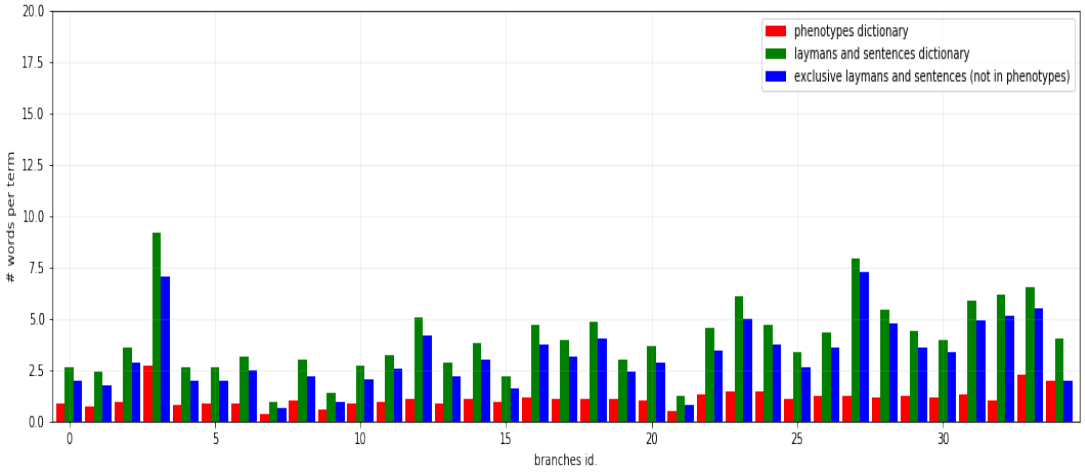
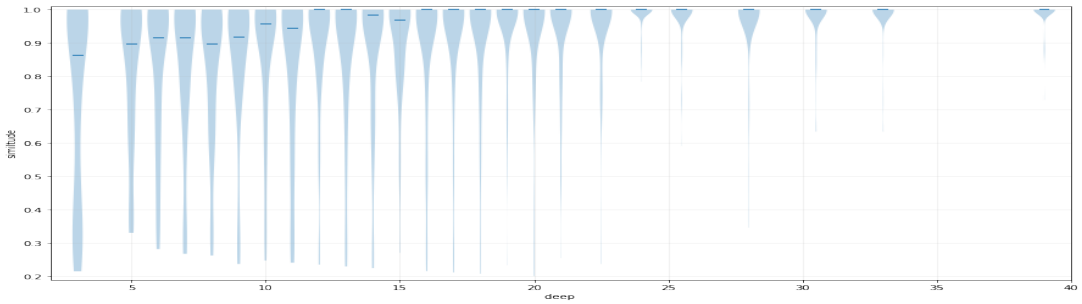
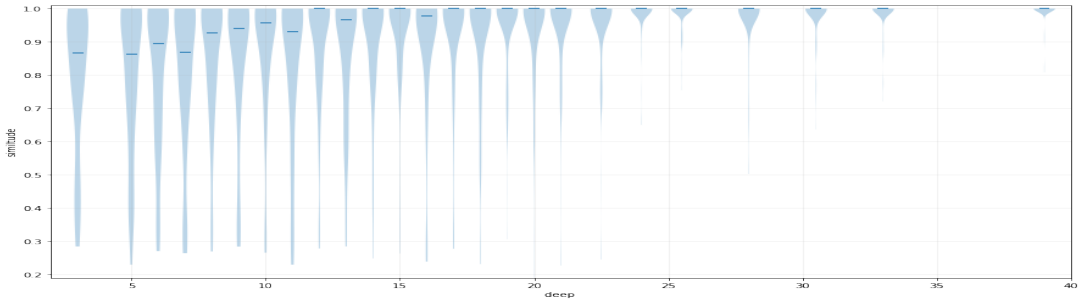


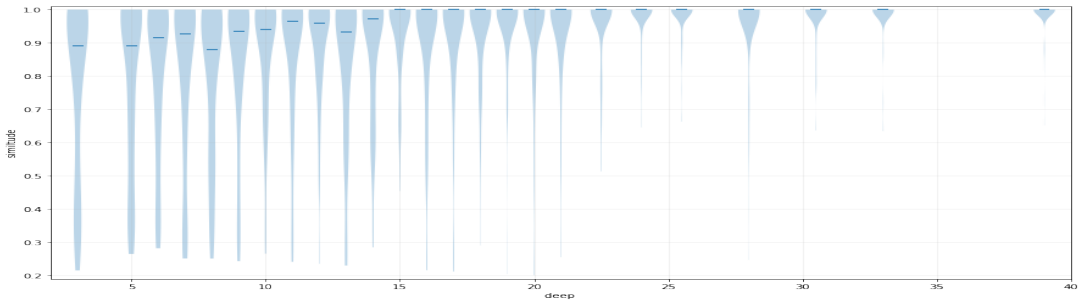
Figure 32: **Distribution of words per term in branches.**



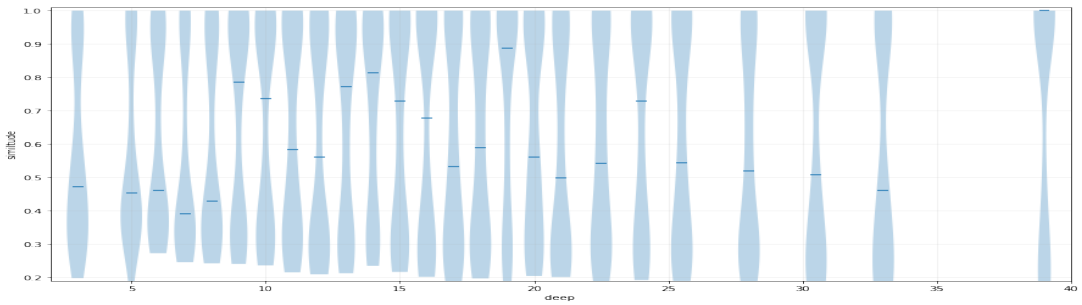
(a) C-LSTM-D(WE_{g1A}, 400, 600)



(b) C-LSTM-D(WE_{lsaB}, 600, 600)

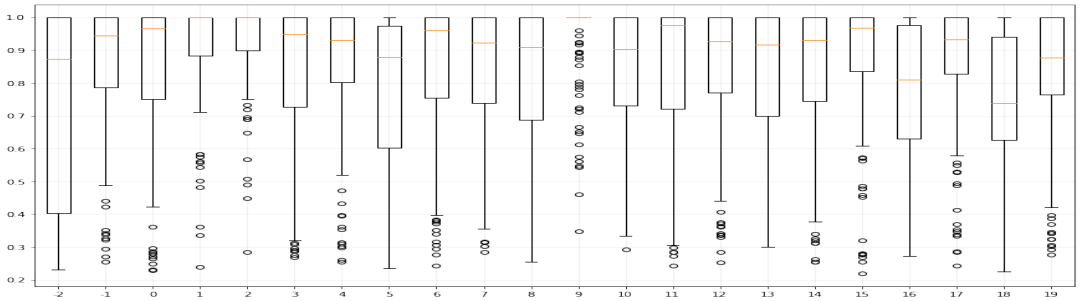


(c) C-LSTM-D(WE_{svdA}, 600, 600)

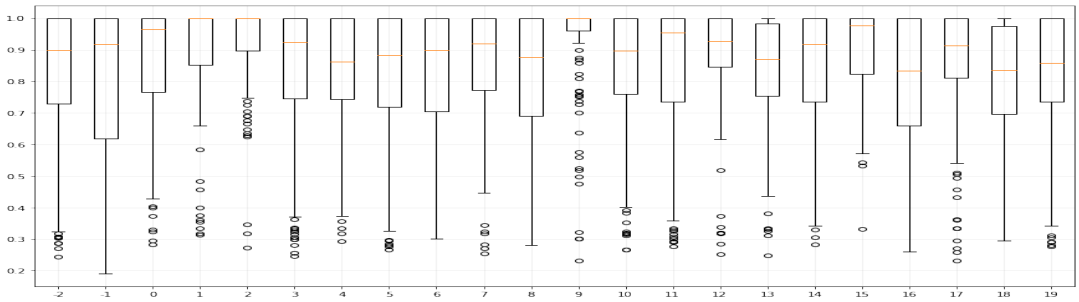


(d) C-LSTM-D(WE_{randA}, 600, 600)

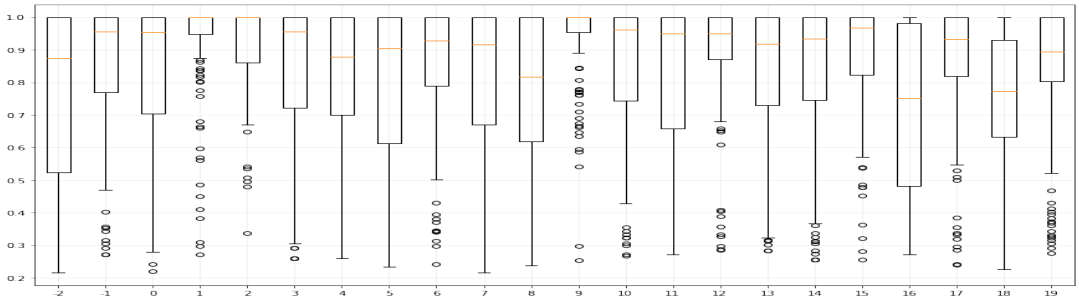
Figure 33: **Predicted similarity per depth level.**



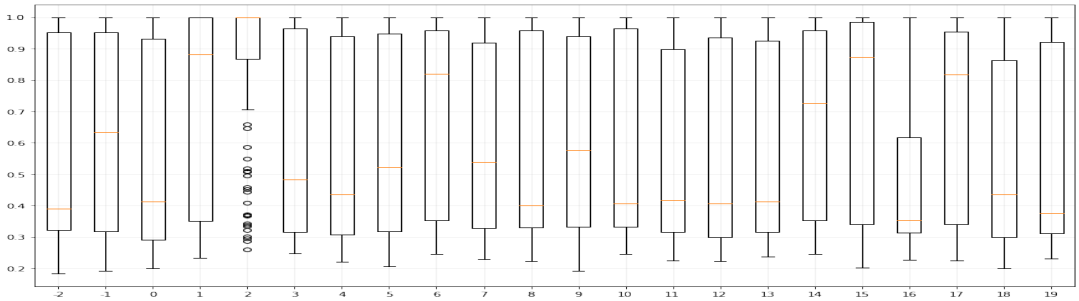
(a) C-LSTM-D(WE_{g1A}, 400, 600)



(b) C-LSTM-D(WE_{lsaB}, 600, 600)

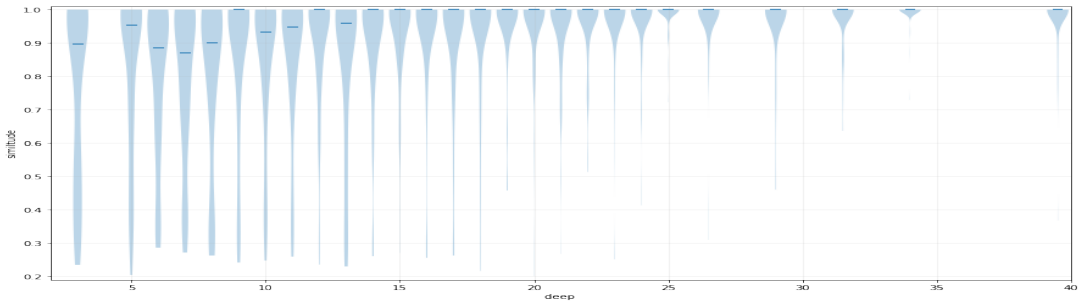


(c) C-LSTM-D(WE_{svdA}, 600, 600)

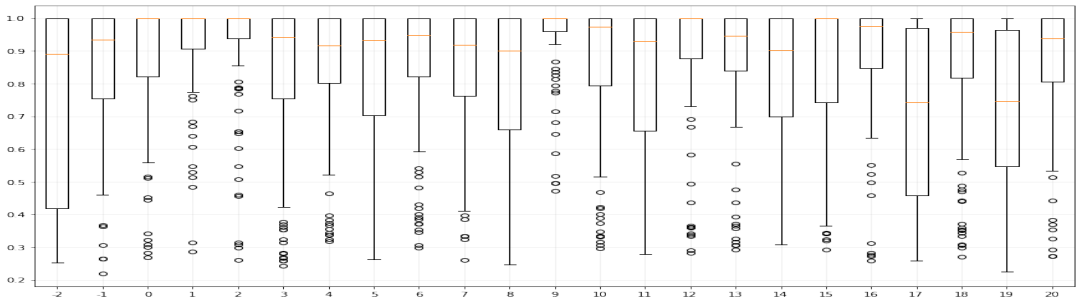


(d) C-LSTM-D(WE_{randA}, 600, 600)

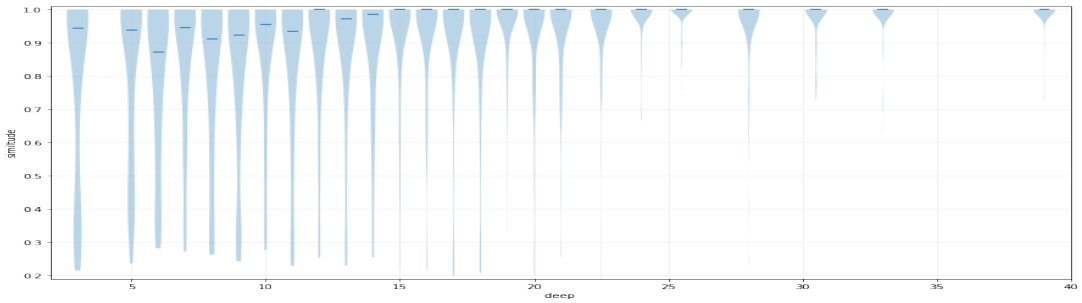
Figure 34: **Predicted similarity per branches.**



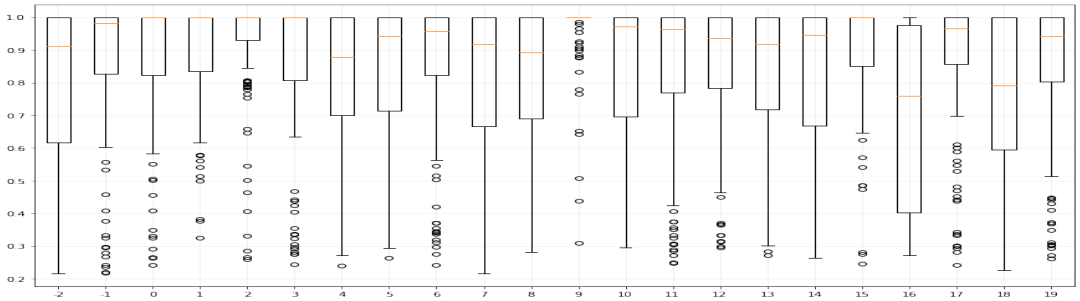
(a) LSTM-D(WE_{g1A}, 400, 600)-Similarity per deep



(b) LSTM-D(WE_{g1A}, 400, 600)-Similarity per branch

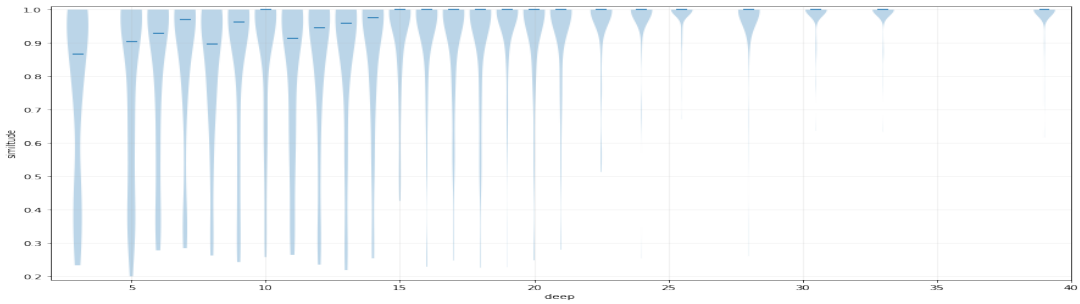


(c) LSTM-D(WE_{svdA}, 600, 600)-Similarity per deep

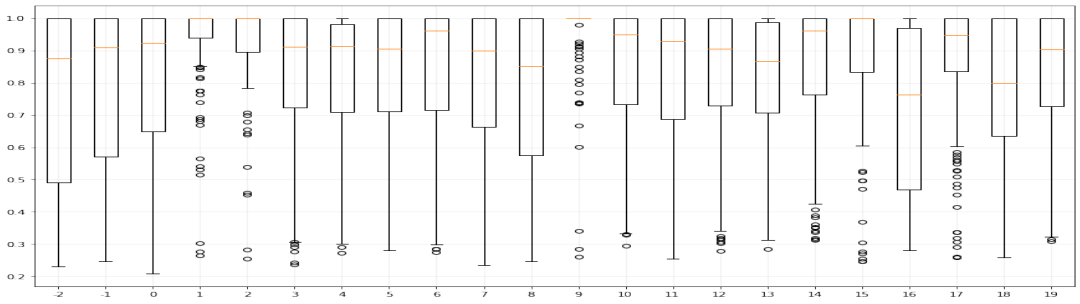


(d) LSTM-D(WE_{svdA}, 600, 600)-Similarity per branch

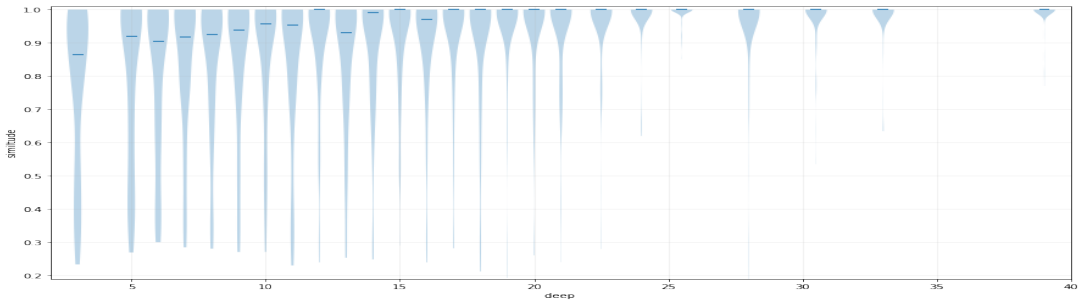
Figure 35: **Predicted similarity per deep or branch for LSTM-D models.**



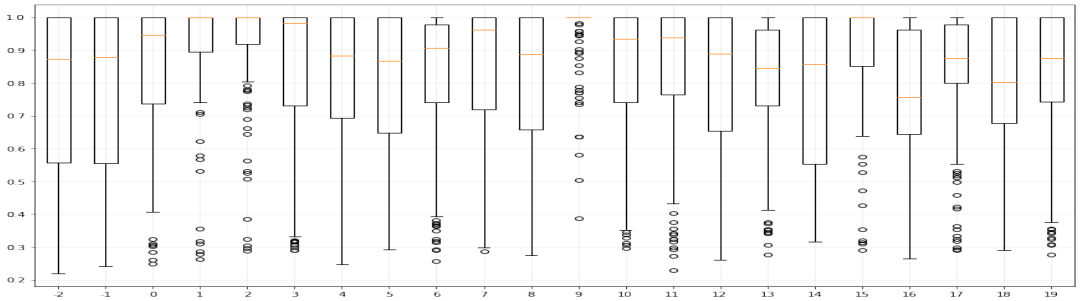
(a) LSTM-P(WE_{g1A}, 600, 600)-Similarity per deep



(b) LSTM-P(WE_{g1A}, 600, 600)-Similarity per branch

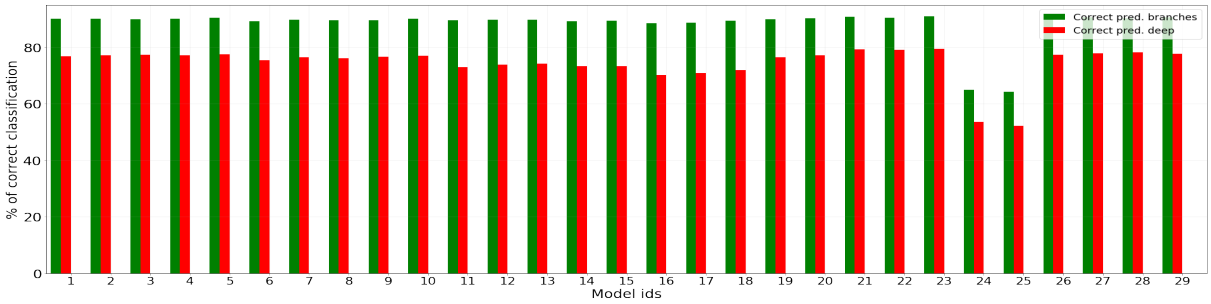


(c) LSTM-P(WE_{lsaB}, 600, 600)-Similarity per deep

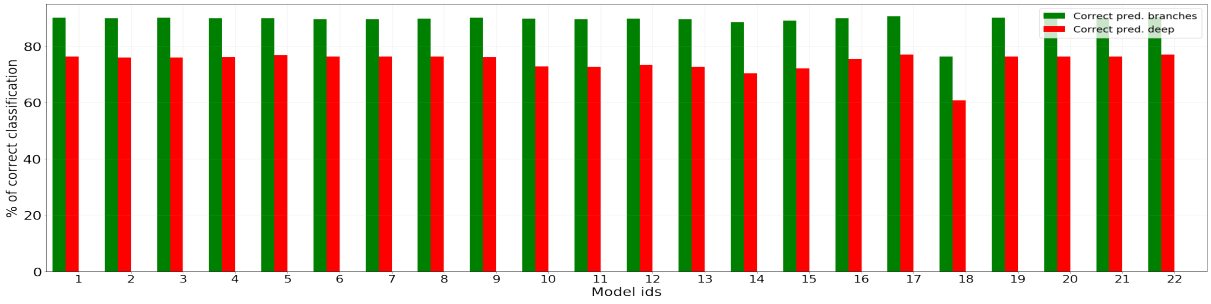


(d) LSTM-P(WE_{lsaB}, 600, 600)-Similarity per branch

Figure 36: **Predicted similarity per deep or branch for LSTM-P models.**



(a) LSTM-D



(b) LSTM-P

Figure 37: **Percentage of correct classification for branches (green) and deep levels (red) per model.**

Appendix D: Python code

```
def create_model_LSTMP(WE_dim, len_WE,N = 400,syn_vocab_size = 13365, syn_max_len = 62):

    # It creates a text classifier model:
    #        INPUT--> WE --> LSTM --> NN dense --> OUTPUT
    # The parameter in input are:
    #      - len_WE = dimension of the initila WE
    #      - N = output dimension of LSTM blocks
    #      - syn_vocab_size = # of words in the input dictionary
    #      - syn_max_len = max len for an input phrases

    model = Sequential()
    model.add(Embedding(syn_vocab_size, len_WE, input_length=syn_max_len, mask_zero=True))
    model.add(LSTM(N))
    model.add(Dense(WE_dim, activation='linear'))
    model.compile(optimizer='adam', loss='mean_squared_error')
    return modellanguage=Python
```

Listing 1: **Create a LSTM-D model**

```
def create_model_CLSTMP(WE_dim, len_WE,  num_filt, D, P, N,K = 0.1, syn_vocab_size = 13365, syn_max_len = 62, pd = 'valid'):

    # Creates a convolutional test classifier model:
    #        INPUT-->WE-->dropout-->Convolutional layer-->max pool.-->LSTM-->NN dense--> OUTPUT
    # The parameter in input are:
    #      - len_WE = dimension of the initila WE
    #      - num_filt = number of filter in concolutional layer
    #      - D = dimension filters
    #      - P = parameter for the max pooling
    #      - N = output dimension of LSTM blocks
    #      - syn_vocab_size = # of words in the input dictionary
    #      - K = dropout parameter (if 0 no dropout is applied)
    #      - syn_max_len = max len for an input phrases
    #      - pd = padding for convolutional layer

    model = Sequential()
    model.add(Embedding(syn_vocab_size, len_WE, input_length=syn_max_len))
    if K != 0:
        model.add(Dropout(K))
    model.add(Conv1D(num_filt, D, activation='relu',padding = pd))
    model.add(MaxPooling1D(pool_size=P))
    model.add(LSTM(N))
    model.add(Dense(WE_dim, activation='linear'))
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

Listing 2: **Create a C-LSTM-D model**

```python
def create_model_LSTMP(WE_dim, len_WE,N,n_branches = 26,syn_vocab_size = 13365, syn_max_len = 62):

    # it creates a text classifier model that takes two inputs, the term to translate and its category.
    # Several Dense layers are put in parallel, each one works for a category:
    #                                                 INPUT 2
    #                                    NN dense       |
    #                                    NN dense       V
    #        INPUT --> WE --> LSTM --> NN dense--> choose one --> OUTPUT
    #                                       ...
    #                                    NN dense
    # The parameter in input are:
    #      - len_WE = dimension of the initila WE
    #      - N = output dimension of LSTM blocks
    #      - N_branches = number of categories (i.e. number of dense layers)
    #      - syn_vocab_size = # of words in the input dictionary
    #      - syn_max_len = max len for an input phrases

    inp_lay = Input(shape = (syn_max_len,))
    inp_branch = Input(shape=(n_branches,))
    emb = Embedding(syn_vocab_size, len_WE, input_length=syn_max_len, mask_zero=True)(inp_lay)
    lstm = LSTM(N)(emb)
    parallel_layer = []
    for i in range(n_branches):
        dense = Dense(WE_dim, activation='linear')(lstm)
        dense = Reshape((1,WE_dim))(dense)
        parallel_layer.append(dense)
    out_dense = concatenate(parallel_layer,axis= 1)
    selected_out = dot([inp_branch,out_dense], axes=1)
    model = Model([inp_lay,inp_branch],selected_out)
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

Listing 3: **Create a LSTM-P model**

```
def clean_dict(dic,stop=False,inp=False):


    # clean the terms in dic by:
    #     - lower all letters
    #     - convert numbers in letter
    #     - separate words divided by a -
    #     - eliminate punctuation and other non printable characters
    #     - if Stop = True, eliminates stopwords
    #     - if inp = true it also separated coupled words (w1-w2) that are not in the list coupled terms and it join the one that are

    cleaned = []
    # prepare regex for char filtering
    re_print = re.compile('[^%s]' % re.escape(string.printable))

    remove = string.punctuation
    remove = remove.replace("-", "") # don't remove hyphens
    table = str.maketrans(remove, ' '*len(remove))
    for line in dic:

        # normalize unicode characters
        line = normalize('NFD', line).encode('ascii', 'ignore')
        line = line.decode('UTF-8')

        # convert to lowercase
        line = line.lower()

        #translate ordinal number (e.g: 1st --> first)
        re_results = re.findall('(\d+(st|nd|rd|th))', line)
        if re_results:
            for enitre_result, suffix in re_results:
                num = int(enitre_result[:-2])
                tmp = num2words(num, ordinal=True)
                tmp += ' '
                line = line.replace(enitre_result, tmp)

        # convert numb-numb couples  numbers to string (e.g.: 2-3 --> two three)
        re_results = re.findall("\d[\d]*[-]\d[\d]*", line) #"[-+]?[.]?[\d]+(?:,\d\d\d)*\d*(?:[eE][-+]?\d+)?"
        if re_results:
            for enitre_result in re_results:
                re_results2 = re.findall('\d[\d]*',enitre_result)
                string_tmp = ' '
                for num in re_results2:
                    tmp = num2words(float(num),ordinal=True)
                    tmp += ' '
                    string_tmp += tmp
                line = re.sub(enitre_result, string_tmp,line)

        # convert al other numbers to string
        re_results = re.findall(" [-+]?[.]?[\d]+(?:,\d\d\d)*[\.]?\d*(?:[eE][-+]?\d+)? ", line)
        if re_results:
            re_results = sorted(re_results, key=len)
            for enitre_result in reversed(re_results):
                try:
                    num = float(enitre_result)
                    tmp = num2words(num)
                    tmp = ' ' + tmp + ' '
```

```
                except:
                    tmp = ' '
                #tmp = tmp[:-1]
                line = line.replace(enitre_result, tmp)

        # eliminate multiple space
        line = re.sub(' +', ' ',line)

        # tokenize on white space
        line = line.split()

        # remove stop words if necessary:
        if stop:
            from nltk.corpus import stopwords
            stop_words = set(stopwords.words('english'))
            stop_words.remove('all') # 'all' in this case is not a stop word
            line = [word for word in line if word not in stop_words]

        # remove punctuation from each token (replace with a space)
        line = [word.translate(table) for word in line]

        # remove non-printable chars form each token
        line = [re_print.sub('', w) for w in line]

        # remove with spaces
        line = ' '.join(line)
        line = re.sub(' +',' ',line)
        line = line.split()

        # look for coupled terms
        if inp:
            line_tmp = line.copy()
            for w in line:
                if '-' in w:
                    if w not in coupled_terms:
                        # the coupled words form two separated words
                        w_clean = re.sub('-',' ',w)
                        w_tok = nltk.word_tokenize(w_clean)
                        line_tmp.remove(w)
                        line_tmp += w_tok

                    else:
                        # the coupled words form one single word
                        w_clean = re.sub('-','',w)
                        w_tok = nltk.word_tokenize(w_clean)
                        line_tmp.remove(w)
                        line_tmp += w_tok

            line = line_tmp

        # store as string
        tmp = ' '.join(line)
        tmp = re.sub(' +',' ',tmp)
        cleaned.append(tmp)

    return array(cleaned)
```

Listing 4: **Implementation of Algorithm 1 for pre-processing**