UNIVERSITY CEU - SAN PABLO

POLYTECHNIC SCHOOL

BIOMEDICAL ENGINEERING DEGREE

BACHELOR THESIS

Development of a classification neural network for the detection of relationships between biomedical entities

Author: Raquel Vázquez Reza
Supervisor: Carlos Oscar Sorzano Sánchez

July 2022

UNIVERSIDAD SAN PABLO-CEU
ESCUELA POLITÉCNICA SUPERIOR
División de Ingeniería

Datos del alumno

NOMBRE:

Datos del Trabajo

TÍTULO DEL PROYECTO:

Tribunal calificador

PRESIDENTE:	FDO.:

SECRETARIO:	FDO.:

VOCAL:	FDO.:

Reunido este tribunal el ______/______/______, acuerda otorgar al Trabajo Fin de Grado presentado por Don________________________________la calificación de _________________.

ACKNOWLEDGMENTS

Al CEU San Pablo y a sus docentes, por acompañarme a lo largo de esta etapa, mostrándose siempre disponibles para ayudarme en mi formación, por su esfuerzo y dedicación hacia los alumnos.

A mi circulo afectivo, amigos y familia, por el apoyo moral y la confianza depositada en mí, sobre todo a mis padres, por el esfuerzo que han mostrado siempre por ofrecerme la mejor educación.

Y a todos los que colaboran en el avance de nuevas soluciones médicas, con el fin de poner a disposición de los sanitarios el mejor equipamiento, ofreciendo a los pacientes seguridad, confianza y calidad.

ABSTRACT

COpenMed is a website created for the structuring of medical knowledge. This website was created to facilitate people's access to medical information. According to the United Nations, more than half of the world's population does not have access to essential health services but more than 60% of the population has access to the Internet. COpenMed is not a diagnostic website, it aims to guide and provide information about symptoms and diseases. The purpose of this Bachelor thesis is to generate, with the use of Python language, a classification neural network to identify if there is a relationship between two biomedical entities registered in the web database. The pre-training of the neural network has been carried out with BERT, for the training of the neural network PyTorch has been chosen. Data on diseases and symptoms were extracted directly from the COpenMed database. For the evaluation of the results, the confusion matrix technique was used. The generated network obtained a 95% and 55% of sensitivity and specificity respectively detecting strong relations between entities.

RESUMEN

COpenMed es una página web creada para la estructuración del conocimiento médico. Esta web fue generada con el fin de facilitar el acceso a las personas a información médica. Según las Naciones Unidas, más de la mitad de la población mundial no tiene acceso a servicios de salud esenciales (pero más del 60% de la población tiene acceso a Internet). COpenMed no es una web diagnosticadora, pretende orientar y facilitar información acerca de síntomas y enfermedades. El propósito de este proyecto es generar, con el uso de lenguaje Python, una red neuronal de clasificación que permita identificar si existe relación entre dos entidades registradas en la base de datos de la web. El preentreno de la red neuronal se ha llevado a cabo con BERT, para el entreno de la red neuronal se ha escogido PyTorch. Los datos sobre las enfermedades y síntomas se han extraído directamente de la base de datos de COpenMed. Para la evaluación de los resultados se ha usado la técnica de matriz de confusión. La red generada ha obtenido un 95% y 55% de sensibilidad y especificidad respectivamente detectando relaciones fuertes entre entidades.

INDEX

FIGURE INDEX

TABLE INDEX

1 INTRODUCTION

1.1 Motivation

The Universal Declaration of Human Rights [1] states that: "Everyone has the right to a standard of living adequate for the health and well-being of himself and of his family...". The current world situation is far from this right, according to World Health Organisation (WHO) [2]: "Half the world lacks access to essential health services...". This means that 50% of the population does not have access to regular health consultations. In contrast, according to the International Telecommunication Union (ITU) [3], which is the United Nations specialised agency for information and communication technologies (ICT), more than 60% of the population has access to the internet.

Health has always been a topic of interest within society. In recent years, due to the COVID-19 pandemic, also called coronavirus, the volume of searches on health issues, symptoms and diseases has increased. Google trend [4], a free and open access tool provided by Google, which allows us to compare the search popularity of various words or phrases, classifies "symptoms" as a "very recurrent" search in different regions worldwide and the word "disease" as a "recurrent" search (see Figure 1).

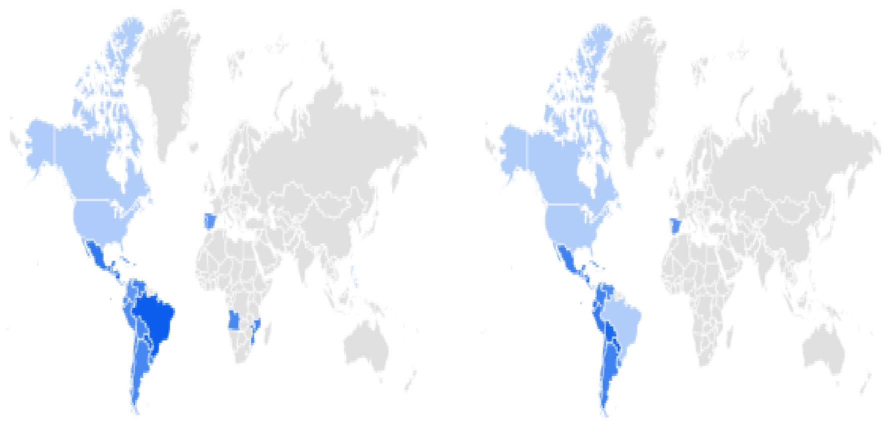


Figure 1*. Google trend detection of searchs 2020-2022. Left picture "symptoms", Right "disease".*

If we enter "cough", one of the symptoms of different respiratory diseases, the search volume expands to many more regions worldwide (see Figure 2).

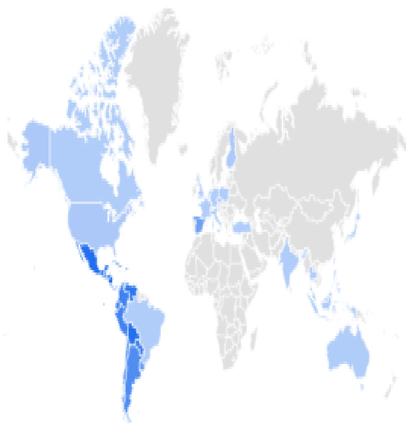


Figure 2. *Google trend detection of searchs 2020-2022."Cough".*

People are interested in knowing the causes of changes in their health, which is why it is essential to provide health information in an appropriate way. CopenMed is an open content website for structuring medical knowledge [5], which aims to offer a web search engine that provides medical information in a correct way, understanding the common language used by users.

Today the amount of information available on diseases, symptoms, causes and treatments in the entire medical field is enormous, far more extensive than a human brain can assimilate. Deep learning techniques are a type of machine learning, artificial intelligence (AI), whose aim is to efficiently automate the process of creating analytical models, allowing to identify and decide as a human brain would do but handling a much larger volume of data. These techniques are composed of artificial neural networks [6], which, as their name suggests, aim to mimic the functioning of neural networks in living organisms. This technique will be explained in depth in section 2.

This Bachelor thesis focuses on the development of a classification neural network, created using Python language, which aims to identify the relationships between different symptoms and diseases, registered in COpenMed [5]. BERT [7] and PyTorch [8] have been chosen for the development of the neural network. Anaconda [9] and Google Colab [10] have been used as working environments.

1.2 COpenMed

COpenMed [5], as previously mentioned, is an open content website for structuring medical knowledge. One can think of this website as a kind of Wikipedia. Is not intended to replace the doctor, nor is it a diagnostic website. The aim of this website is to bring structured health information to everyone.

This website is operated by volunteers and trainees who add contributions to the structuring of the content. Currently, the website has about 10,000 entities and relationships. Each entity is equivalent to a symptom, disease, treatment, or cause. Within the database, the relationships that exist between these entities are established.

The objectives of this website are twofold: to facilitate people's access to medical information using a more commonly used language, but without losing the technical and scientific rigor, and to facilitate the access of machines to this information to build automatic medical "reasoners" that, from the symptoms, can find the most probable causes that provoke them.

This report aims to contribute to the objectives of the creation of this website, building a neural network of classification that is able to identify the relationships between the different entities. This network will contribute to the development of the web search engine that will be generated for the website.

1.3 Software

1.3.1 Anaconda

Anaconda [9] is the world’s most popular open-source Python distribution platform. This open-source software has been choosing because is intuitive, allowing easy installation of packages. The packages are securely hosted and regularly updated. The software has been installed for use under the SO of Windows and allows the use of various packages used in Machine Learning techniques (e.g. TensorFlow, torch...).

Spyder [11] is a powerful scientific Python development environment pre-installed in Anaconda. It includes editing, interactive testing, debugging, and introspection features. During this Bachelor thesis the 5.0.1 version has been used.

Python is a programming language that prioritises code readability. Versions 3.9 and 3.7 have been used. Python code is readable and consistent furthermore the libraries provided by Python are diverse, containing many data science and mathematics-oriented libraries. One of the most prominent libraries for developing Deep Learning techniques and provided by Python is TensorFlow [12], which is an open-source library developed by Google, used to build and train neural networks.

1.3.2 Google Colab

Google Colab [10] is a Google tool known as "Colaboratory", this tool allows programming, running and sharing code, using the Python language, within Google Drive. Colab can be thought of as a JupiterNotebooks notebook stored in Google Drive, this notebook is connected to a "runtime" in the Colaboratory cloud so it allows to run Python code without prior installation. During execution, Colab offers a specific amount of RAM and Hard Disk space, if this space is exceeded the code stops executing. The code can be shared via Google Drive or Github. Google Colab notebooks are stored in the same format as Jupiter notebooks "ipybn". This programme has been selected as a method for developing and executing code, because it is particularly suitable for machine learning tasks. Google Colab provides free access to GPUs, is always ready to use and requires no installation of libraries, which makes it easier and faster to run code.

1.3.3 BERT

To improve the classification performance of the neural network, a pre-training of the network has been developed. Neural networks are composed of different layers, in each of which there are a different number of neurons. The neurons in the network have a numerical weight, which modifies the received input, which must be numerical.

At the beginning of the training of a neural network [6], the weights accompanying the neurons are usually initialised randomly, these weights will be adjusted by mathematical functions until an optimal output is obtained.

A BERT [7] pre-training stage allows the weights of the network to be easily adjusted when the network is trying to obtain an answer from the language expressed by humans. BERT pre-training finds a good set of vector representation weights to compress the input data and usually achieves better network classification performance because it provides a numerical representation of the expressed sentence that can be processed by the neural network.

BERT [7], Bidirectional Encoder Representations from Transformers, is an AI-based system. The system was developed to assist Google Search in natural language understanding (NLP) tasks. In this way, Google can better understand the language commonly used by people. COpenMed [5] aims to bring medical information closer to the user by using a more common language, so the objectives that BERT fulfils are in line with what is desired on this website.

BERT is bidirectional, as its name suggests, because it analyses the sentence in two directions. This task is carried out using a novel technique called masked LM (MLM). It analyses the words before and after a keyword. This property allows it to deeply understand the context of phrases. It prioritises search intent over keywords.

This system has been used as a pre-training method for the classification neural network developed during this Bachelor thesis. The BERT code has been modified using Spyder, Python 3.7, to suit the intended purpose. The code has then been run using Google Colab to employ the use of GPUs. BERT tools will be explained in section 2.3.1, BERT implementation will be described in section 3.1.

1.3.4 PyTorch

PyTorch [8] is an open-source library with a great ease of use, it is created to be flexible. This library has a syntax and application similar to Python, its documentation is organized and useful for beginners, it is simple to use, the learning curve for developers is short. It is focused on performing numerical calculations using tensor programming, this property facilitates its use in deep learning tasks.

PyTorch employs data parallelism which allows it to distribute the work among multiple CPU or GPU cores. The use of GPUs accelerates the training of models. PyTorch has been a key player in the development of relevant artificial intelligence applications, such as Tesla's Autopilot.

These features make PyTorch one of the most affordable options for building neural networks. This library is included in Anaconda. PyTorch tools will be explained in section 2.3.2, PyTorch implementation will be described in section 3.2

1.4 Phases and Structure of the memorie

Throughout this report, the process of creating, executing and evaluating the pre-training and training of the classification neural network will be described.

The paper is divided into the following sections: section 2, describes the principles of neural networks and the mathematical procedures of Bert and PyTorch, section 3, describes the database used and the process of creating the pre-training and training of the neural network, section 4, describes the evaluation tests carried out and presents the results obtained from the training, section 5, discusses the results, and section 6, presents conclusions drawn from this Bachelor thesis.

2 ARTIFICIAL NEURAL NETWORKS

2.1 Introduction

The brain is an organ of the human body, [13]. It is made up of more than a billion neurons. Neurons are a type of cell in the nervous system. They work together, giving us the ability to reason and understand the world.

Biologically [14], a neuron is made up of: The body of the cell, or soma, which houses the nucleus. Dendrites, which are branches that start from this cell and form a dense network, and the axon. The axon is a longer fibre, usually about one centimetre long, which allows connections to be made between different neurons (see Figure 3). The connection established between neurons is called a synapse. The number of connections established by a neuron varies from a dozen to a thousand.

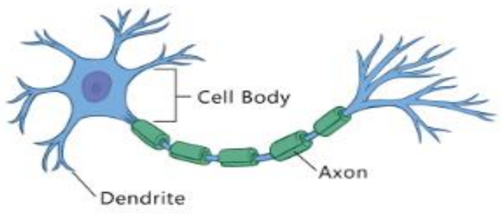


Figure 3. *Neuron anatomy.*

Neurons communicate by means of signals [14]. Signals are propagated by electrochemical reactions. The synapse, or connection established between neurons, releases transmitter chemicals that enter through the dendrites, raising the electrical potential of the cell body (see Figure 4). If the set limit is reached and exceeded, an electrical impulse, also called an action potential, is sent to the axon. The impulse diffuses down the branches of the axon, eventually reaching the synapse and releasing transmitters into the bodies of other neurons. There are two types of synapses, excitatory, which increase the potential, or inhibitory, which decrease the potential. Another point to note is the plasticity of synaptic connections. Plasticity is a property that allows the intensity of connections to be altered over time in response to a simulated pattern, the brain learns from the experience. These mechanisms are the foundation of learning in the brain.

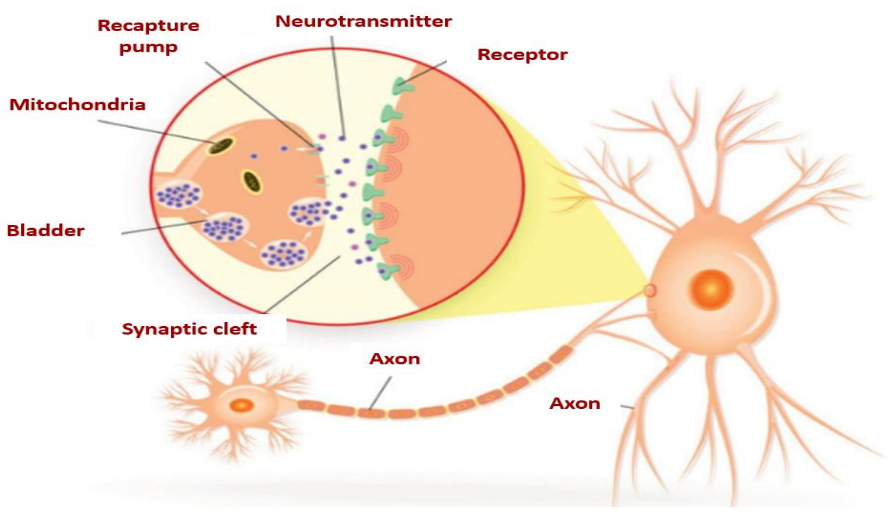


Figure 4. *Neural signal transmision.*

People have a great capacity for reasoning. We can perform several operations simultaneously, we have a very complex, non-linear and parallel information management system. Our brain, however, has some limitations. Many learning tasks require a large analysis of a huge amount of data to draw conclusions, our brain has a limited memory for analysis compared to computers. Computers allow us to store an enormous amount of data, they also speed up the results, surpassing the human brain in speed. However, in recognition tasks, the human brain still outperforms the computer.

Brains and computers can resemble. The brain picks up signals from the environment, processes the information it receives and generates a response from, as discussed in the previous section, electrical impulses and chemical reactions. Computers contain microprocessors. Microprocessors treat information in the form of electricity, process it based on their established programming and generate an output response.

Artificial intelligence systems aim to take advantage of both brain and computer systems. This is the reason for the creation of neural networks**.**

2.2 Components and biological comparisons

The following section explains the components of artificial neural networks and their similarities to the biological ones they are inspired by.

The key element of a biological neural network is the neuron, likewise, the key element of an artificial neural network is the "node", also called neuron. The nodes are linked by connections and are organized in layers. We can classify the layers as: external layers and hidden layers. The external layers can be of two types, input layers and output layers. Input layers are composed of nodes that are connected to an external environment; they receive external stimuli. Output layers, give response to the system. The hidden layers are composed of nodes that establish connections with other nodes, they have no contact with the outside. Each node, as biological neurons, can receive “n” inputs since they can stablish different connections (see Figure 5).

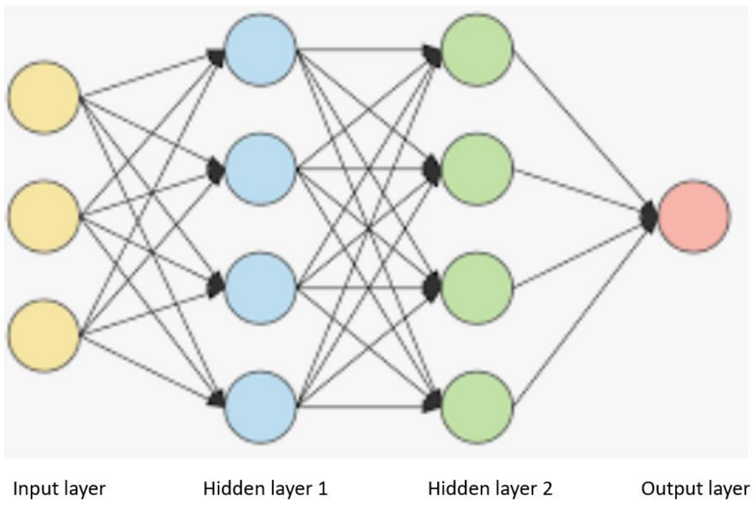


Figure 5. *Neural Nework Scheme.*

Each connection established between nodes has assigned a numerical weight, “wij”. This is the main long-term memory resource. In the previous section, it has been argued that neurons can have either inhibitory or excitatory power, artificial neurons can also have this power. If the weight "wij" has a positive value, it indicates that the relationship between neurons, as in biological neurons, is excitatory, this means that if neuron “i” is activated, neuron “j” will receive a signal that will tend to activate it. If, on the contrary, the weight "wij" has a negative value, the relationship between neurons will be inhibitory, if neuron “i” is activated, it will send a signal that will deactivate neuron "j". The learning that the neural network performs during its training is done by updating these weights. This property is similar to the plasticity shown by the brain when learning new things, commented in section 2.1.

The "wij" weights measure the intensity of interaction between the nodes that are connected. At node "i", the sum of the “n” inputs "xj", weighted with the weights "wij", generates the total input or "post-synaptic potential" of node "i". The activation function “f” is the one that will designate the output "yi" of the node, to this function is applied the difference between the "postsynaptic potential" and the threshold “Ɵi”. If the threshold is exceeded, a connection with the next neuron is established (see Figure 6).

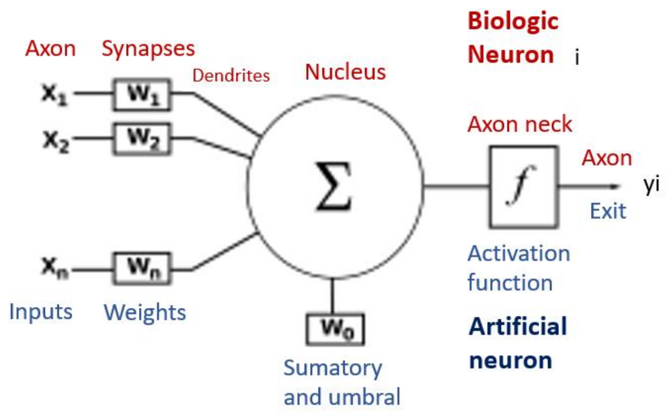


Figure 6. *Artificial and biological neuron components.*

Biologically, it is generally accepted that the information stored in the brain is more related to the synaptic values of the connections between neurons than to the neurons themselves. Knowledge is found in the synapses. Likewise, in artificial neural networks, knowledge is found in the "wij" weights. Every learning process involves a certain number of changes in these connections.

2.3 Inside Characteristics

This section will describe the configuration characteristics an the mathematical inside of the neural network developed during this Bachelor thesis.

Neural networks are composed, as mentioned in previous sections, of layers containing nodes. The nodes establish connections between them. The connectivity between the nodes of a network is related to the way in which the outputs of neurons are channeled to become inputs to other neurons. There are two types of connectivities forward and backward. Forward propagation fed the input data in the forward direction through the network. Backward propagation allows cycles and connections with previous levels (see Figure 7). PyTorch [8] and BERT [7] make use of both, after performing forward propagation, they perform backward propagation to update the weights.

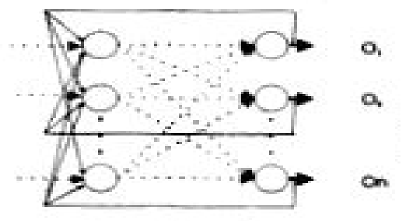


***Figure 7**. Back propagation network.*

There are different types of learning, supervised and unsupervised learning. BERT use an unsupervised learning, the data introduced is not labelled. The type of learning used during the training part, developed using PyTorch, is supervised learning. This learning makes use of labels, where the machine is provided with a set of labeled data so that it knows how to draw conclusions and classify from these labels. This learning method has been used because the existing relationships between entities are available.

A binary classifier has been used for the application of this learning. Binary classifiers distinguish between two categories, "related" 1 or "unrelated" 0. This classifier will make use of linear and non-linear functions.

The size of the network must be chosen carefully for the network to be optimal. A network size that is too large will present generalization problems when entries are introduced that he has not previously seen. A network size that is too small will not be able to represent the desired function. This problem makes it necessary to be careful with the size of the network you choose. Following sections will explain more in detail the logistics inside BERT and the neural network developed using PyTorch.

2.3.1 BERT

BERT [7] is conceptually simple and empirically powerful. It consists of two steps: pre-training and fine-tuning. During this work we have made only use of the pre-training which will provide us with a numerical representation of the input that will facilitate the training of the neural network.

During pre-training, the model is trained on unlabeled data, phrases composed of biomedical entities and relationships, using two unsupervised tasks. Task one, Masked LM and task two, Next Sentence Predicition (NSP). MLM task mask some percentage of the input tokens at random, and then predict those masked tokens. The final hidden vectors corresponding to the mask tokens are fed into an output sofmax over vocabulary. In this Bachelor thesis we masked over 15% at random. The masked words will be predicted later with cross entropy loss.

The input of this model must be a sentence. Bert will mask this sentence during MLM tasks. This technique uses a token vocabulary [15], where we can find: [CLS] which stands for a special classificaton token and is placed at the beginning of the sentence. [SEP] that separates sentences and it is placed at the end of sentences. [UNK] which refers to unkown token, [MASK] which replaces the keywords from which we will draw context and [PAD] that is used to adjust the sentence to the size you have specified the input sentences should be, if the sentence is shorter than expected, [PAD] is added to reach the desired length.

BERT uses Transformer during pre-training for obtaining numerical representation with which to work mathematically. Transformer is a mechanism capable of learning the contextual relationships between words. Transformer offers different mechanisms. BERT uses the Transformer mechanism as an encoder that reads the input text and converts it into a vector.

Transformer reads the entire sequence of words at once, which is why it is considered bidirectional, it learns the context of a word based on its entire environment (left and right). This method uses as input a sequence of tokens, these are converted into vectors and then processed in the neural network. The output is a sequence of vectors where each vector correspond to a token.

Transformer as an encoder has two layers: self-attention and feedback. Self-attention is the method that allows to reformulate the representation of a word, based on all the other words in the sentence.

The Softmax [16] activation function is the activation function of this model. This function converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector (see Figure 8).

$$softmax(Z_i) = \frac{\exp(Z_i)}{\sum \exp(Z_i)}$$

***Figure 8**. Softmax activation function.*

Cross-entropy loss [17] is used when adjusting model weights during training. The aim is to minimize the loss, the smaller the loss the better the model. A perfect model has a cross-entropy loss of 0. Cross-entropy function is defined as H(P,Q), where P is the target distribution, and Q is the approximation of the target distribution (see Figure 9).

$$H(P, Q) = -\text{sum } x \text{ in } X\ P(x) * \log(Q(x))$$

***Figure 9.** Cross-entropy function.*

The purpose of the Cross-Entropy is to take the output probabilities (P) and measure the distance from the truth values (see Figure 10). The error formula, showed in figure 10, applies the algorithm in such a way that the closer the output is to 1, the smaller the error, the closer the output is to 0, the higher the error.

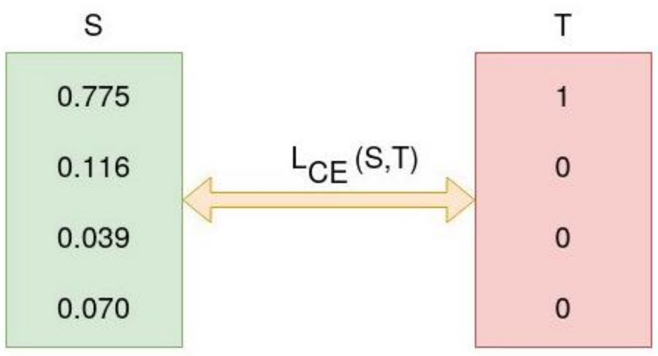


***Figure 10.** Distance between left and right values.*

NSP task is not used in this pre-training since we are only generaiting one sentence per entity. This pre-training part is the most expensive part of the computing, Once the Bert neural network has been trained following the previous steps, we can obtain the embeddings of each entity, which will be the contextual numerical representations of each token, that will serve as input for the training of the classification neural network developed with PyTorch.

2.3.2 PyTorch

PyTorch [18] is a machine learning library that make compatible usability and speed: it provides an imperative and Pythonic programming style that supports code as a model, makes debugging easy and is consistent with other popular scientific computing libraries, while remaining efficient and supporting hardware accelerators such as GPUs.

The PyTorch neural network receives as input a data table that must contain labels. The labels serve to identify the result of our classification. To avoid overtraining the neural network, the initial data is divided into test and train. Train refers to the data that will be used to train the network. Test data is the one not used during training to avoid overfitting. The PyTorch neural network will process the data in batch mode. This library follows the following network schema shown in Figure 11:

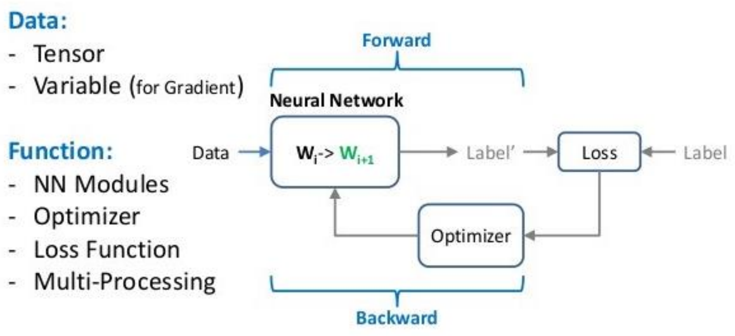


Figure 11. *PyTorch neural network scheme.*

In the neural network, data and weights are entered. The output of the activation function will be the weighted sum of the previous input (see Figure 12), where "b" is bias, "x" is the input data, "w" refers to the weights. The network is composed of different linear layers. Each linear layer is followed by a ReLu function, a batch normalization and a dropout function. The linear layers will contain the nodes.

ReLu function is a rectification function for applying nonlinear functions. Batch normalization is a method that normalizes each batch of data, prevents very different distances between data. Dropout function is a regularization technique to reduce the overfit, this technique takes some weights and puts them at 0.

$$z = b + \sum_{i=0}^{n} wi + xi$$

Figure 12. *Activation function.*

Subsequently, the function loss of the output "z" and the expected output "y" is calculated, using BCE with logits loss. This function combines a Sigmoid layer and the BCELoss in one single class. Sigmoid applies the element-wise function (see Figure 13). BCE Loss creates a criterion that measures the Binary Cross Entropy between the target and the input probabilities (see Figure 14), where "n" is the batch size. The final result will be the mean of L.

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + \exp(-x)}$$

Figure 13. *Sigmoid Function.*

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_n \left[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n) \right],$$

Figure 14. *BCE loss function.*

Once the final output is obtained, we proceed to the last step, the optimization of the weights. This optimization is carried out by Adam optimizer. Adam optimizer involves a combination of two gradient descent methodologies, momentum and root mean square propagation. Momentum accelerates the gradient descent algorithm by computing the exponentially weighted average. The use of averages makes the algorithm converge towards the minima in a faster pace. Root mean square propagation takes the exponential moving average of the squared gradients. Adam inherits the strengths of the positive attributes of these two methods and build a more optimized gradient descent (see Figure 15).

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta w_t}\right] v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta w_t}\right]^2$$

Figure 15. *Adam optimizer. Left side of the equation equals to momentum. Right side of the equation equals to Root mean square propagation.*

Adam optimizer gives much higher performance than other optimizers (see Figure 16), the graph shows how this optimizer outperforms the rest in terms of training cost (low) and performance (high). Therefore it has been selected for his use.

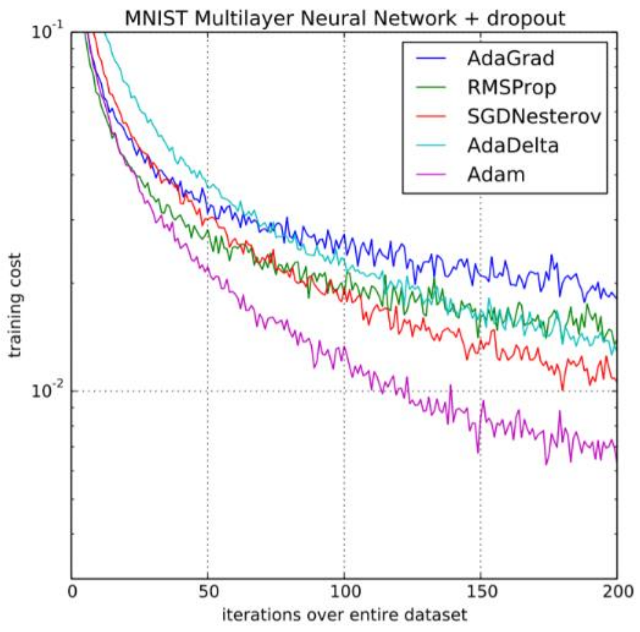


Figure 16. *Training cost vs performance of optimizers graph.*

This library includes different modules. These modules allow PyTorch to calculate the gradients of the weights of the different layers at the same time as it applies the forward phase to the train data. Other machine learning techniques, such as Keras, calculate the gradients statically and then apply the algorithm.

Another advantage of PyTorch is the use of tensors. Tensors, as mentioned before, are multidimensional matrices. The tensors used by PyTorch require an indication of whether the gradient needs to be calculated for the particular tensor or not.

3 MATERIALS AND METHODS

3.1 Input Files Bert pre-training Obtention

COpenMed [5] database has been used during this report. This database currently contains 10,000 entities. The database also contains the relationships between these entities. The entities contained are different diseases, causes, symptoms and treatments. All the data recorded in this database have been entered by volunteers or students and supervised by the site leaders.

Recorded data can be represented using graph theory. The world today can be represented in graphs to understand it almost fully. Graph models for database management are extremely powerful and allow us to know information that is hidden from the naked eye. A graph [19] is a mathematical structure that allows us to model everyday problems through a graphical representation formed by nodes or vertices and edges. In this case, each node will be an entity of the database, and each edge will represent the links or relationships between these entities. On the COpenMed website [5], if we access an entity, we will be able to visualize a star graph, where there is a central node and directed edges connecting with other nodes (see Figure 17).

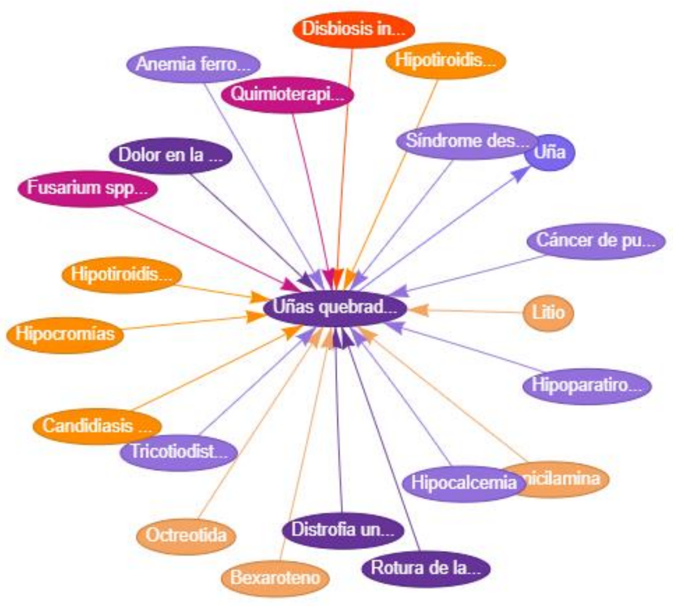


Figure 17. *Directed star network of the entity brittle nails.*

The distance between the central node and the external nodes indicates the closeness of the relationship between these entities, the closer the central node is to an external node, the closer the relationship is. In the database, the level of relationship between entities is also established numerically in a range of 1-0, where "1" designates a very strong relationship and "0" a non-existent relationship. The colors designate the type of node type: symptom, anatomy, treatment, etc. (see Figure 17).

Each of the nodes with which the central node connects, which is the entity we have searched for, contains another star graph, so that from an entity/node "a" we can reach an entity/node "c", with which there is no direct edge/relationship, through an entity/node "b" with which “a” and “c” separately establish a direct relationship. To better visualize this example, from the entity brittle nails (“Uñas quebradizas”) shown in Figure 17 we can reach through hypothyroidism (“Hipotiroidismo”) the entity fatigue (“cansancio”) (see Figure 18).

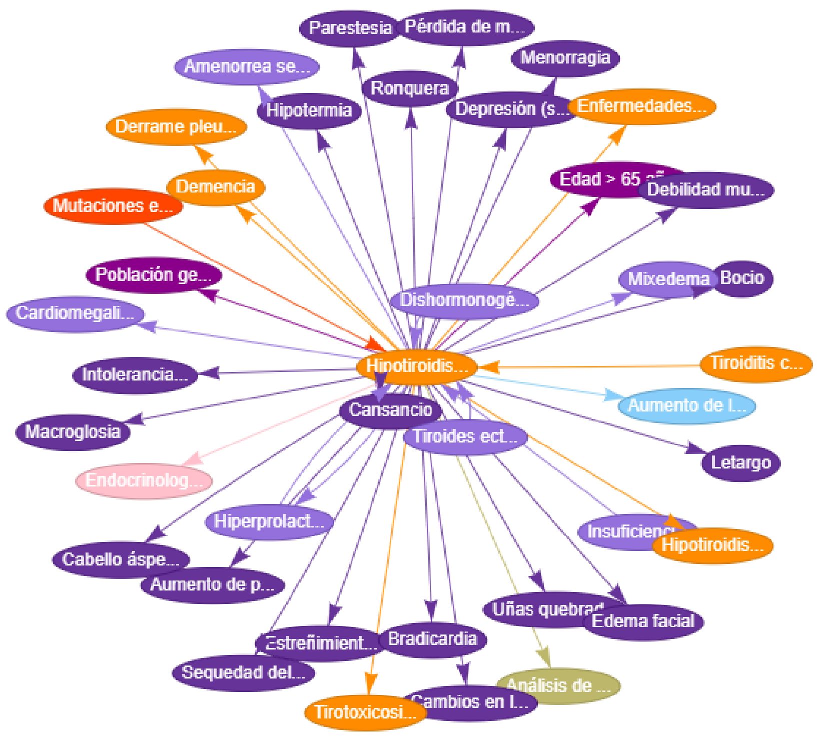


Figure 18. *Directed star network of the entity hypotyroidism.*

If we were to observe a complete graph of the database where all the entities and all the relationships were observed, (see Figure 19), we would be able to observe how from one entity we can reach another through various nodes, on this theory are based the random walks that will be explained below and that as mentioned before, allow us to see connections established between entities, which at first sight cannot be observed.

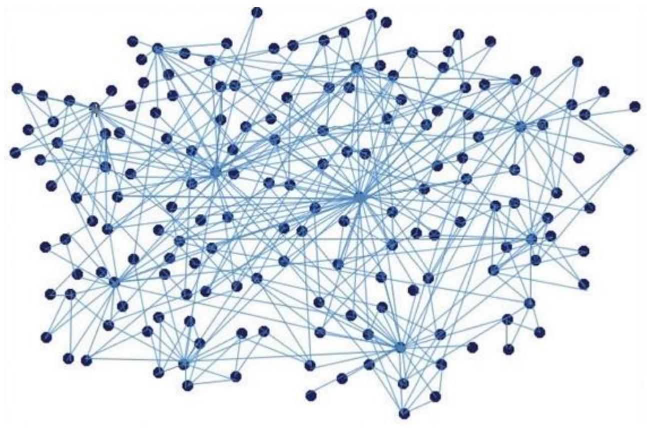


Figure 19. *Global network example.*

The objective of the neural network to be developed is to detect relationships between entities. BERT is the first step to be developed for this purpose. This pre-training phase requires a plane text file, containing one sentence per line. To facilitate this work, input files have been generated, one file for each entity, "randomWalkxxx_walks.txt", where "xxx" is the number of the entity. Each file contains 100 random walks starting from the selected entity, the walks have a length of 30 entities. Each walk will be equivalent to one sentence. The random walks start from the selected entity and connect to different nodes (entities), to which they are related. The random walk stops when it reaches 30 different entities.

The paths will have a total length of 61 words, 30 entities, 30 relationships and a word <eol> indicating the end of the path. Entities will be represented by an "e" followed by the entity number; relationships will be represented by an "r" followed by the relationship number (see Figure 20).

```
e9 r321 e641 r321 e4193 r8 e614 r77 e6394 r77 e6422 r16 e5504 r16 e6197
```

Figure 20*. Random walk example from entitie 9.*

BERT also uses a text file called vocabulary, mentioned in section 2.3.1. This vocabulary, contained in "randomWalk_vocab.txt", is used to know the possible words, contained in the sentences, that we are going to introduce. To generate this text file, the possible special tokens that will be introduced by the MLM model: [PAD], [UNK], [CLS], [MASK], [SEP], and the possible entities and relations contained in the sentences have been introduced in it. The vocabulary consists of a total of 9381 words.

The special tokens are intended to be a representation. Each token has a different meaning, the meanings have been described in section 2.3.1. Common words are not used because we are training a model that uses words. These tags only have the purpose of being a sentence-level representation for classification.

3.2 Bert Pre-training implementation

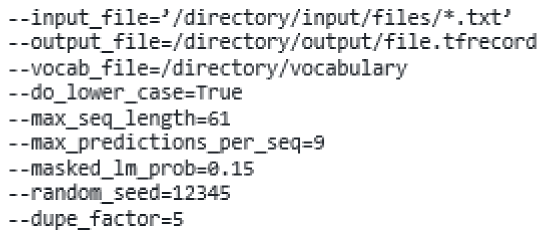
For the development of the code used for the pre-training of the neural network, we have resorted to a Github repository, where the BERT code is published [20].

In this dissertation only the pre-training part will be used. The pre-training of the neural network has a high cost of time and CPU, GPU or TPU usage, but it is a one-time procedure. The code we will use is an adaptation of BERT Python, the original BERT code was written in C++.

To facilitate the use of the code in the future, a Colab file containing 5 cells has been generated, each code cell is in charge of one step of the pre-training. This full code, and all the files necessary to reproduce it are in a repository of Github [21].

In the first cell we find the code needed to connect to our Google Drive. It is necessary to connect with Google Drive to obtain the input_files, the code and indicate where we want to generate the output_files. In this cell we also indicate the versions of the libraries we want to use. In this case, we will use Tensorflow 1 and numpy 1.19. These versions are necessary because the code has been developed on them. The Python version that manages this libraries version is Python 3.7.

In the second cell we find the first step to start the pre-training, "create_pretraining_data". This execution file needs the determination of some inputs (see Figure 21). "--input_file" refers to the directory where the text files that contains the random walks are located, "--output_file" refers to the location where we want to place our output, this file must be of TFRecord extension, "vocab_file" refers to the location of the file containing the vocabulary, "do_lower_case=True "is used to convert all letters to lowercase and not interpret them differently, "-max_seq_length "is used to indicate the maximum length of the sentences we want, in our case, the length will be 61, "masked_lm_prob" is used to indicate the number of words to be masked, "random_seed" is used to indicate the seed we want to randomly start from and "dupe_factor".



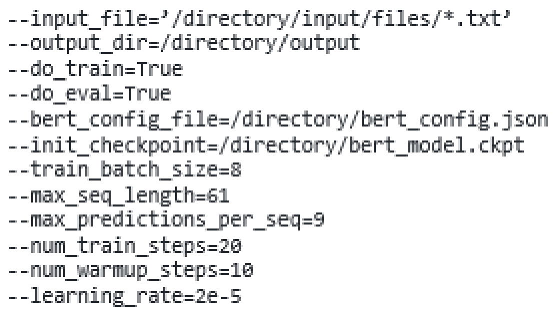
```
--input_file='/directory/input/files/*.txt'
--output_file=/directory/output/file.tfrecord
--vocab_file=/directory/vocabulary
--do_lower_case=True
--max_seq_length=61
--max_predictions_per_seq=9
--masked_lm_prob=0.15
--random_seed=12345
--dupe_factor=5
```

Figure 21. *Create pre-training data inputs.*

Create_pretraining_data will generate the input needed to perform the pre-training. To achieve this goal, the text files containing the random paths are read, and once the sentences containing the random paths are obtained, 15% of the words in each file are randomly masked with the available tokens. The output file that will be generated during the execution will contain the masked sentences, the position where the masking took place and the word that previously occupied the place where the mask is located. This file has an extension of type Tfrecord. Tfrecord is a binary file consisting of a sequence of records in which each record is a string of bytes, allows more efficient storage, takes up less space than the original data and can be split into several files. This type of file allows the data to be stored unordered, without losing the order, which fulfils the objective to be obtained at the exit of the execution.

In the third cell, we have the code needed to perform the pre-training. Before running this step, we need to modify the execution environment that Google Colab is using. For this, the offered graphical interface is used and we will select GPU as the execution environment. This execution file needs the specification of some inputs (see Figure 22). "--input_file" as we have explained before, the input file corresponds to the

Tf_record file generated in the previous step. "--output_dir" is the directory where we want to generate our output, “--do_train” and “--do_eval” we will set them to true to get results, “--bert_config_file”, we will specify the location of the bert_config_json file, this file contains different specifications, size of the vocabulary, how many sequences are masked, the size of the embeddings... "--init_checkoint" will determine the location of the file bert_model.ckpt, "--train_batch_size" refers to the size we want to take from all the elements, to train them together, "max_seq_Length", has the same meaning as before and must have the same value. "max_predictions", must be the result of multiplying the previous parameter by the masking probability.



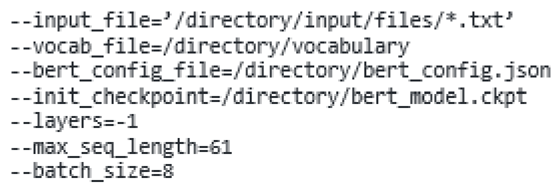
```
--input_file='/directory/input/files/*.txt'
--output_dir=/directory/output
--do_train=True
--do_eval=True
--bert_config_file=/directory/bert_config.json
--init_checkpoint=/directory/bert_model.ckpt
--train_batch_size=8
--max_seq_length=61
--max_predictions_per_seq=9
--num_train_steps=20
--num_warmup_steps=10
--learning_rate=2e-5
```

Figure 22. *Run pre-training inputs.*

This code aims to obtain some initial weights, using the Tfrecord generated in the previous section as input. This code file is intended to learn how to unmask the masked words, for which it uses the previously explained cross-loss function. The output of this execution is the result of the probability with which it unmasks the words in a correct way, it also generates a series of files that mark the weights with which the final tests have been generated.

The last cell contains the code necessary to execute “extrac_features”. This is the last step of our pre-training, in this step we will obtain the pre-trained contextual embeddings of each entity in the database. These embeddings are fixed contextual representations of each input token generated from the hidden layers of the pre-trained model. The code has been modified to generate one numpy file per entity, and it has also been specified that this numpy file must contain the embeddings generated for the token [SEP], located in the last layer, layer -1, because it contains the representation of the entire sentence. BERT is designed, as previously mentioned, not only to perform

MLM tasks but also to perform NPL tasks. In this work only MLM tasks are used, that is why the code areas where NPL is used have been removed.



```
--input_file='/directory/input/files/*.txt'
--vocab_file=/directory/vocabulary
--bert_config_file=/directory/bert_config.json
--init_checkpoint=/directory/bert_model.ckpt
--layers=-1
--max_seq_length=61
--batch_size=8
```

Figure 23. *Extract features inputs.*

The necessary inputs (see Figure 23) are: "--input_file" we will specify the path to our text file containing the generated sentences, "randomWalk_walks.txt", "--output_file", we will specify the path where we want to deposit our output file, this file will be transient since it will be read later and modified to generate an .npy per entity. "--vocab_file" specify the path to the file containing the vocabulary, "--bert_config", "--maxseqlength" and "--batch_size" are the same as in the previous steps, "--init_chekpoint" specify the path containing the check points file generated in the previous step, "--layers", indicates the number of layers to which we want to access, in our case it will only be necessary to access layer -1.

3.3 Input files PyTorch obtention

The pre-training performed with BERT resulted in a .npy file for each entity in the database. This file contains the pre-trained contextual embeddings for each entity for the token [SEP].

The input file necessary for the training to be carried out with PyTorch must be made up of a column of type 'label', which indicates whether or not there is a relationship between two embeddings, and two other columns that will each contain an embedding of an entity.

Different text files have been generated to prevent memory collapse during execution. A Python code has been generated that reads the existing relationships between the entities. For each relationship, 10 lines of text are generated, containing a "1" marking the existing relationship between the two entities, and where a random embedding is taken from each of the entities that make up the relationship. After writing these lines, another 10 lines of text are written, containing a "0" marking the non-relationship between two entities, and where a random embedding is taken from one of the related entities and a random embedding is taken from another entity unrelated to the previous entity. The files contain 16,000 lines each.

This text files are intended to ensure that there are enough existing or non-existing relationships to train the neural network. This code and the code that has been developed to train and test the PyTorch neural network has been developed in Spyder 5.1.0, Python 3.9. Google Colab has not been used for space reasons.

3.4 PyTorch training implementation

The implementation of the classification neural network developed during this Bachelor thesis makes use of different libraries, among which we can highlight torch and sklearn.

Torch is the library that allows us to make use of PyTorch, a tool presented in section 1.3.4 and whose inner workings have been presented in section 2.3.2. The Sklearn library is used to make analytical predictions, which help us to evaluate the training of the network.

When developing a neural network, we must be clear about its purpose before proceeding with its design. In this case, as mentioned before, the neural network will perform a binary classification and its inputs will be two embeddings, one of each entity. This already gives us the necessary input and output neurons, as mentioned in section 2.3.2 the network must contain an adequate number of neurons for the learning to be correct. The number of neurons and linear layers chosen for this neural network is as follows: The embeddings are formed by 768 inputs each, each neuron will take care of one input, so the total number of input neurons is 1536, the network is composed by two hidden layers of neurons, the first one is composed by 512 neurons and the second one by 128 neurons, finally the output layer is composed by a single neuron that will mark if there is an existing relationship or not between the two inputted entities, this output will be marked with a "1" if there is an existing relationship or by a "0" if there is not an existing relationship. Different operations are necessary between these layers, which have been presented in section 2.3.2 and help the operation of the network (see Figure 24).

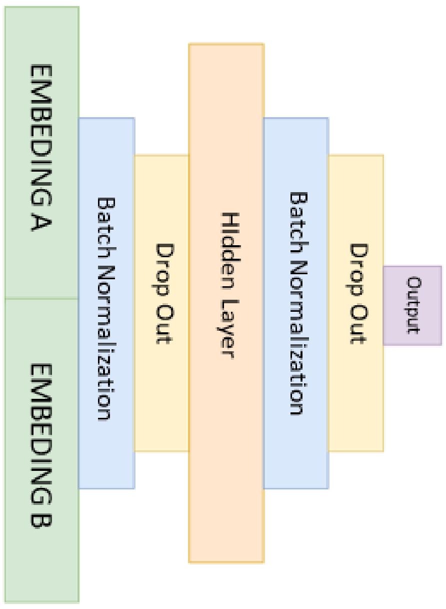
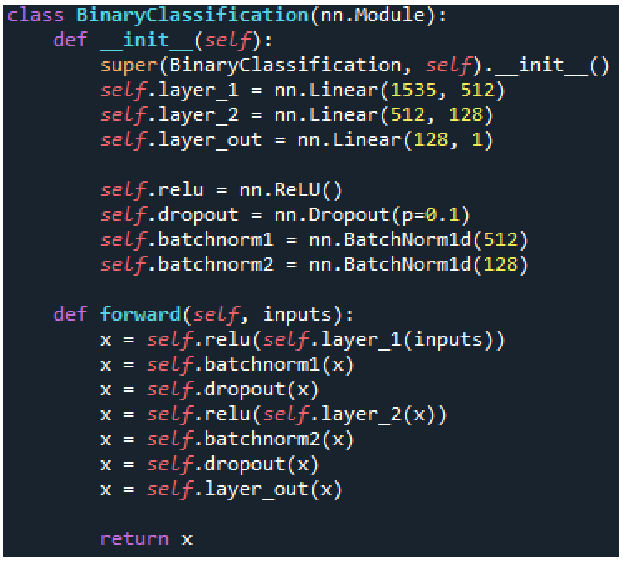


Figure 24. *Neural network final scheme.*

The implementation of this scheme and its directionality has been done as follows (see Figure 25). As can be seen in the image below, a declaration of the layers and functions to be used is made. Subsequently, the order of these is declared, where the "x" containing the entered entries goes through the established order.



```
class BinaryClassification(nn.Module):
    def __init__(self):
        super(BinaryClassification, self).__init__()
        self.layer_1 = nn.Linear(1535, 512)
        self.layer_2 = nn.Linear(512, 128)
        self.layer_out = nn.Linear(128, 1)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=0.1)
        self.batchnorm1 = nn.BatchNorm1d(512)
        self.batchnorm2 = nn.BatchNorm1d(128)

    def forward(self, inputs):
        x = self.relu(self.layer_1(inputs))
        x = self.batchnorm1(x)
        x = self.dropout(x)
        x = self.relu(self.layer_2(x))
        x = self.batchnorm2(x)
        x = self.dropout(x)
        x = self.layer_out(x)

        return x
```

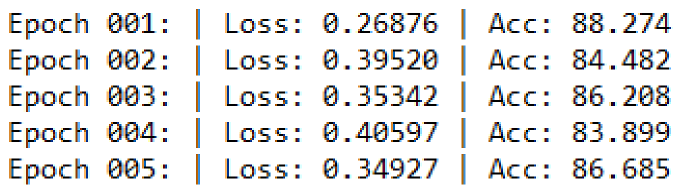
Figure 25. *Neural Network structure declaration.*

To train the network correctly, it is necessary to make several passes through the neural network to adjust the weights, this is called backward propagation. During backward propagation a sigmoid layer is also applied in the BCEWithLogisticLoss function. To fulfil this purpose, the training of the network will be carried out in epochs, where in each epoch there will be different batches. The batches indicate how the input data is divided to perform several passes through it. The epochs mark how many times the training is to be repeated. As mentioned in section 3.3 different input files have been obtained, in total 90 files, to perform the training 1800 epochs have been declared, in each epoch a different file will be chosen which will be divided in batches. To avoid overlearning of the network, every 90 epochs the files are reordered randomly so that they are not always entered in the same order**.**

To do this, two nested for loops have been designed, the outer one will increment the epochs, the inner one the batch used. Before starting the loop that will perform the network training, it is necessary to declare the optimisation and loss functions that will be used. As mentioned and explained in section 2.3.2 these functions are BCEWithLogisticLoss and Adam optimizer. The learning rate, which is the exchange rate at which the weights are updated, used has been set as 0.001.

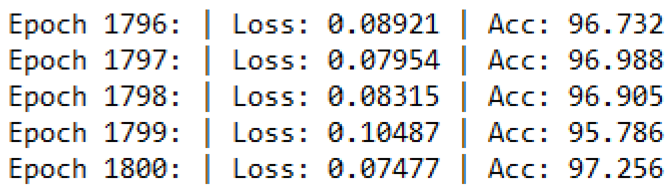
Once we have all the parameters declared we start the for loop mentioned above. This loop will start executing in epochs, in each epoch, it will load a data file with which it will train the network, divide it into batches and start the training carried out by the internal for loop. In each batch it passes the data through the neural network, calculates the obtained loss and the accuracy and performs a summatory of both parameters separately. It carries out the optimisation step where it seeks to improve the assigned weights and cleans the gradient so that it does not accumulate.

To observe the progress of the network, the loss and the accuracy obtained at each epoch has been displayed on screen and recorded in a text file (see Figure 26 and Figure 27). To calculate the final loss and accuracy for each epoch, both are divided by the number of batches used. As can be seen the Loss in the latter epochs (see Figure 27) is much closer to zero than in the earlier epochs (see Figure 26) and the Accuracy in the latter epochs (see Figure 27) is higher than in the earlier epochs (see Figure 26).



```
Epoch 001: | Loss: 0.26876 | Acc: 88.274
Epoch 002: | Loss: 0.39520 | Acc: 84.482
Epoch 003: | Loss: 0.35342 | Acc: 86.208
Epoch 004: | Loss: 0.40597 | Acc: 83.899
Epoch 005: | Loss: 0.34927 | Acc: 86.685
```

Figure 26. *Loss and Accuracy of the first 5 epochs.*



```
Epoch 1796: | Loss: 0.08921 | Acc: 96.732
Epoch 1797: | Loss: 0.07954 | Acc: 96.988
Epoch 1798: | Loss: 0.08315 | Acc: 96.905
Epoch 1799: | Loss: 0.10487 | Acc: 95.786
Epoch 1800: | Loss: 0.07477 | Acc: 97.256
```

Figure 27. *Loss and Accuracy of the last 5 epochs.*

Once the model has been trained, the weights obtained in the last epoch are saved in a file in order to be able to use the final network at another time without having to re-train it.

4 RESULTS

The effectiveness of the neural network in identifying whether two entities are related has been evaluated using the confusion matrix technique.

A confusion matrix [22] is a tool that allows the visualisation of the performance of an algorithm designed for supervised learning tasks. It consists of columns and rows, in the columns we find the possible predictions, in the rows the actual classification. From this matrix we can observe how many times the algorithm has correctly identified whether there is an existing or non-existing relationship, between two entities, true positive (TP) and true negative (TN), and how many times it has been wrong, false positive (FP) and false negative (FN), (see Figure 28).

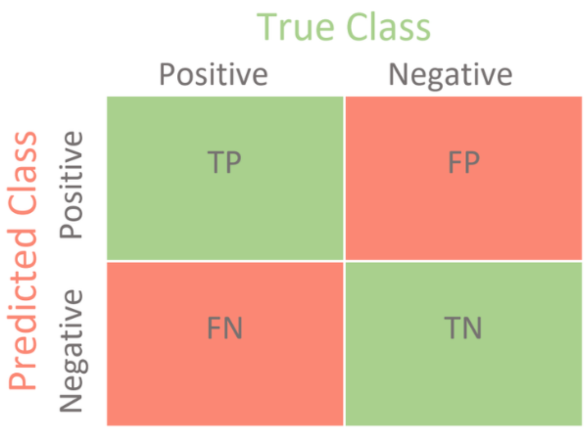


Figure 28. *Confusion matrix structure.*

The effectiveness of the network classification has been evaluated under two different scenarios. The first scenario contains 100 pairs of strongly related entities, relationship level "1", and 100 pairs of unrelated entities. The second scenario contains 100 pairs of entities with a relationship level of "0.5", and 100 pairs of unrelated entities. A numpy file has been generated for each of the scenarios.

Once the files were obtained, the trained neural network model was loaded and the data chosen for the test were passed through the neural network. Finally, the results obtained by the neural network were compared with the real results, generating a confusion matrix that will allow the model to be evaluated. This process was repeated for each of the scenarios (see Table 1 and 2).

	TRUE	
PREDICTED	Related	Unrelated
Related	96	50
Unrelated	4	50

Table 1. *Confusion Matrix level 1.*

	TRUE	
PREDICTED	Related	Unrelated
Related	96	50
Unrelated	4	50

Table 2. *Confusion Matrix level 0.5.*

From the confusion matrix we can obtain many measures that will allow a correct evaluation of the model, among these we find the precision of the model, the recall, the f1-score and the accuracy. Precision [23] is the ability of an instrument to give the same result for different measurements under the same conditions (see Figure 29).

$$Precision = \frac{Tp}{Tp + Fp}$$

Figure 29*. Precision formula.*

Recall [23] allows to observe the sensitivity and specificity of the model, sensitivity characterises the capacity of the neural network to detect existing relationships between entities, specificity indicates the ability of our neural network to identify non-existing relationships. Accuracy is the ability of an instrument to give the desired result accurately (see Figure 30).

$$a)\ Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

$$b)\ Sensitivity = \frac{TP}{TP+FN}$$

$$c)\ Specificity = \frac{TN}{TN+FP}$$

Figure 30. *Accuracy, Sensitivity and Specificity formulas*.

A model can be very precise and not accurate or very accurate and not precise (see Figure 31). The best quality scientific observations are both accurate and precise.

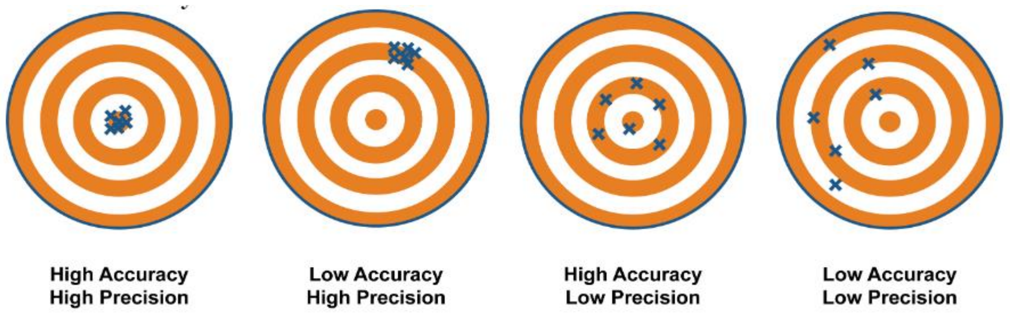


Figure 31. *Accuracy and Precision comparision*.

The F1 value [23] combines the precision and recall measures into a single value. This is practical because it makes it easier to compare the combined performance of accuracy and completeness between various solutions. It is calculated by making the harmonic measure between the two parameters (see Figure 32).

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Figure 32. *F1-score formula.*

The values obtained from these measurements from the confusion matrixes shown were the following (see Table 3 and Table 4*).*

	Precision	Recall	F1-score
Related	68%	95%	79%
Unrelated	92%	55%	69%
Accuracy	-	-	75%
Macro avg	80%	75%	74%
Weighted avg	80%	75%	74%

Table 3. *Precision, Recall, F1-score and Accuracy obtained, level 1.*

	Precision	Recall	F1-score
Related	66%	96%	78%
Unrelated	93%	50%	65%
Accuracy	-	-	73%
Macro avg	79%	73%	71%
Weighted avg	79%	73%	71%

Table 4. *Precision, Recall, F1-score and Accuracy obtained, level 0.5.*

5 DISCUSSION

The results obtained show that the network has a greater capacity to identify existing relationships, sensitivity 95% for level 1 relationships and 96% for level 0.5 relationships, than for identify non-existing relationships, specificity 55% for level 1 and 50% for level 0.5. While it shows a higher precision in identifying non-existing relationships, 92% for level 1 and 93% for level 0.5, than for identify existing relationships, 68% for level 1 and 66% for level 0.5.

The entities, as mentioned in previous sections, establish many connections with each other, being able to reach very distant entities with a low level of relationship with the initial entity, following random paths. The neural network has been trained with relationships of all levels. The network has not been trained to directly identify non-existent relationships, which is why the precision of non-existent relationships can be so high, 92% precision for level 1 and 93% precision for level 0.5, it classifies everything it does not identify, in the same area. As there are relationships at level 0.1, which is a very low level, the network's ability to identify non-existent relationships, level 0, is affected, with a specificity of 55% for level 1 and 50% for level 0.5.

Regarding the identification of relationships, the network shows a high ability to identify strong and medium relationships, with a sensitivity of 95% for level 1 and 96% for level 0.5. Again, being trained on all types of relationships, strong and non-strong, it makes sense that it is able to identify strong and medium relationships well. At the same time, it is normal that the precision of the network is lower, 68% for level 1 and 66% for level 0.5, as the network has not been trained on only one type of relationship between entities, not all relationships between entities will be located in the same area.

The results of the network in the two scenarios do not differ much from each other, reaching 50% specificity and 95% sensitivity in both cases. The behaviour of the network does not vary much when it comes to identifying relationships of different levels.

6 CONCLUSIONS

The final objective of this final degree project was to develop a classification neural network capable of detecting existing relationships between medical entities. The classification neural network developed has a good ability to detect relationships between entities, reaching 95% and 96% of detection for relationships with level 1 and 0.5 respectively. On the other hand, the detection of non-existent relationships is lower, but still reaches or exceeds 50%: 55% and 50% detection for level 1 and level 0.5 respectively.

As future lines of work, it would be interesting to study the possibility of teaching the neural network when two entities are not related in order to increase its specificity, or the differences between related entities at level 0.1 and unrelated entities. These goals can be achieved by changing the type of data with which the network is trained. On the other hand, it is necessary to repeat the process for new entities, introduced in the database. As new objectives to extend the capabilities of the search engine, it would be convenient to continue with BERT's lines of work as well as to look for new lines of work for the development of the neural network in order to rule out better development possibilities.

7 REFERENCES

[1] Web page of the United Nations Human Rights https://www.ohchr.org/en/special-procedures/sr-health/international-standards-right-physical-and-mental-health#:~:text=Everyone%20has%20the%20right%20to,age%20or%20other%20lack%20of (Accessed: April 2022).

[2] Web page of the World Health Organisation https://www.who.int/news/item/13-12-2017-world-bank-and-who-half-the-world-lacks-access-to-essential-health-services-100-million-still-pushed-into-extreme-poverty-because-of-health-expenses (Accessed: April 2022).

[3] Web page of the International Telecommunication Union https://www.itu.int/es/mediacentre/Pages/PR-2021-11-29-FactsFigures.aspx (Acessed: April 2022).

[4] Web page of Google Trend https://trends.google.es/trends/?geo=ES(Accessed: May 2022).

[5] Web page of COpenMed https://copenmed.org/#/paginaInicio (Accessed April 2022).

[6] Web page of Neural Networks Description https://www.cienciadedatos.net/documentos/py35-redes-neuronales-python.html (Accessed: May 2022).

[7] Vaswani A, Shazeer N, Parman N, Uszkoreit, Jones L, Gomez A, Kaiser L, Polosukhin I, Attention is all you need, *31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA, April 2019.

[8] Web page of PyTorch https://pytorch.org/ (Accessed: April 2022).

[9] Web page of Anaconda https://anaconda.cloud/ (Accessed: April 2022).

[10] Web page of Google Colab https://colab.research.google.com/ (Accessed: April 2022).

[11] Web page of Spyder https://www.spyder-ide.org/ (Accessed: April 2022).

[12] Web page of TensorFlow https://www.tensorflow.org/learn?hl=es-419 (Accessed: April 2022).

[13] Web page of Medline Plus https://medlineplus.gov/spanish/ency/anatomyvideos/000016.htm (Accesed: May 2022).

[14] Web page of Neural networks fundaments https://thales.cica.es/rd/Recursos/rd98/TecInfo/07/capitulo2.html (Accessed: May 2022).

[15] Web page of Github BERT Tokens https://albertauyeung.github.io/2020/06/19/bert-tokenization.html/ (Accessed: May 2022).

[16] Web page of softmax activation function https://vidyasheela.com/post/softmax-activation-function-in-neural-network-formula-included (Accessed: May 2022).

[17] Web page with Cross Entropy Loss info https://medium.com/unpackai/cross-entropy-loss-in-ml-d9f22fc11fe0 (Accessed: May 2022).

[18] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Köpf A, Yang E, De Vito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S, Pytorch: An imperative style, high-performance deep learning library, *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, Vancoucer, Canada.

[19] Web page with graph information https://www.grapheverywhere.com/grafos-que-son-tipos-orden-y-herramientas-de-visualizacion/ (Accessed: June 2022).

[20] Web page with pre-training Bert information https://github.com/google-research/bert (Accesed: September 2021).

[21] Web page with pre-training Bert repository code in Github https://github.com/cossorzano/copenmed_tools (Posted: June 2022).

[22] Web page with Confusion Matrix information https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/ (Accesed: June 2022).

[23] Web page with precision , recall and f1-score information https://www.iartificial.net/precision-recall-f1-accuracy-en-clasificacion/ (Accessed: June 2022).