caption

Technical University of Catalonia

Master in Innovation and Research in Informatics
Data Science

Question Classification in the Medical Domain using Convolutional Neural Networks and Word Embeddings

Student
Alexandra Yamaui

Supervisors
Lluís Belanche
Horacio Rodríguez

October 15, 2018

Contents

Abstract

Question Answering Systems have become important due to the necessity to retrieve answers from questions stated in Natural Language, without the need of querying structured data sources. One of the key steps of these systems is the question classification process, where a label with the type of the expected answer is assigned to every question. Even though a lot of research has been conducted for open domain question answering systems, more specialized models are still needed for certain fields, such as the field of medicine, that could be shown as one of the most important. In this project we build a question classification model using biomedical questions from the BioASQ challenge 2018 and use its predefined classes. Additionally, we introduce other more specific categories that aim to describe the topic of the question. We use a simple model based on a Convolutional Neural Network and dense vector representations for the questions to obtain a classifier that has good overall performance on predicting the type of questions and a better capability on detecting their topics. We have also added co-supervised labelled questions from an external source.

1 | Introduction

Question Answering (QA) can be defined as the task of, given a question expressed in Natural Language (NL), providing to the user the correct answer of the question, unlike as in the Information Retrieval systems in which a set of documents where the answer is likely to be found is retrieved. Although the origin of QA can be found in the eighties of the last century, with the development of NL Interfaces to the computer applications, especially to the databases (CHAT-80 (Warren and Pereira, 1982) is one of the best-known Natural Language Interface of the early eighties), the term QA was started to be used in the framework of QA tracks within the Text Retrieval Conferences, TREC challenges[1], starting with TREC-8[2] in 1999.

Usually, users prefer to be given the answer to their information needs without the necessity of querying structured data sources. For this reason, the role of Question Answering Systems (QAS) in current technology has become more and more important due to its ability to satisfy the necessity of providing advanced and user-friendly search tools.

A lot of improvement has been done in open-domain QAS, however, domain-specific QA systems are still needed in order to improve their accuracy with specialized information. One of the things that influence the performance of a QAS the most is the question classification step (Hovy et al., 2001). Question classification consists of assigning a label or a category to a question, i.e. the Question Type (QT), to determine what type of answer is required, i.e. the Expected Answer Type (EAT). For that reason, this work is focused on developing a classification model that can successfully classify biomedical questions with little hyperparameter tuning.

We used as resource the 2018 BioASQ[3] challenge (Cohen et al., 2017). BioASQ is a platform that organizes challenges on biomedical semantic indexing and question answering. The tasks include hierarchical text classification, machine learning, information retrieval and QA from texts. We will focus on the latter one, doing question classification using the 4 general categories provided in the challenge, whose description is shown in the section 4.1. Additionally, we manually created another 9 categories whose aim is to describe the topic or content of the question (see section 4.1).

[1] http://trec.nist.gov

[2] http://trec.nist.gov/data/qa/t8_qadata.html).

[3] `http://www.bioasq.org`

Traditional approaches to solve question answering problems are based on matching the question with a set of patterns or rules that reflect its syntactic structure. Usually, these patterns are built manually, which despite the good results that it provides, is inconvenient and not agile. Moreover, although there are works that try to create these patterns automatically ((Biswas P, 2014), (Xu J, 2012), and (Mourad Sarrouti, 2017)), neural models have shown a superior performance with little hyperparameter tuning. For this reason, we decided to use Convolutional Neural Networks (CNN) as classification method. Additionally, in light of the good results obtained by the use of dense vectors for text representation, we decided to build our own embedding vectors at word and sentence level using the Word2Vec model (Mikolov et al., 2013b). We built these vectors using the syntactic structure of the questions and the underlying relations between words inside a sentence (see section 4.2). We also used alternative pre-trained embeddings, namely the well recognized GloVe vectors (Pennington et al., 2014) and Pyysalo biomedical embeddings (Moen and Ananiadou, 2013).

In our implementation, we performed two types of classifications. The first, only using the 4 original classes in the challenge, which we called the *Multiclass* escenario, and the second, where we predict a label for the general class and a label for the specific category, which we called the *Multiclasss-Multilabel* setting. In a supplementary way, we developed two mechanisms (described in section 4.1) to increase the number of samples including (initially unlabeled) questions of an external data set.

Conducting this work, we succeeded on building a model with high predicting performance using GloVe pre-trained vectors, a simple CNN and dependency relations between words. However, it is fair to mention that the model does not have the same prediction capability for all the classes. We also made improvements in accuracy using the extra questions labeled with one of the mechanisms we developed to increase the number of samples. Lastly, we discovered that the model is suitable for topic extraction tasks.

The structure of the document starts with the State Of The Art in Chapter 2, where we introduce general concepts about question answering and the most relevant and recent approaches. Then in Chapter 3 (Background) we explain the concepts that are directly related to our implementation, including Convolutional Neural Networks and embedding vectors. Chapter 4 is the one that contains the main part of the work we did, presenting the corpus used (section 4.1), the techniques we applied to encode the questions (4.2) and the architecture we built to classify them (4.3). After that, we will explain the experiments and results obtained in chapter 5 and finally, we will present the conclusions and future work in the last chapter (6).

1.1 Objectives of the thesis

As we mentioned before, our approach is to use a CNN and word and sentence embeddings to build a biomedical question classifier. In summary, the specific objectives are:

- Build word and sentence embedding vectors using the Word2Vec model and the syntactic and lexical structure of the questions.
- Classify questions into the 4 general classes (Multiclass) using a convolutional neural network.
- Classify questions into the 4 general classes and the 9 specific classes (Multiclass-Multilabel) using a convolutional neural network.
- Compare the performance of the model using n-hot encoding vectors against word embedding and sentence embedding vectors.
- Compare the performance of the word embeddings against the GloVe and Pyysalo pre-trained vectors.
- Compare the model with a simple linear model using n-hot encoded vectors.

2 | State Of The Art

2.1 A short review of Question Answering

QA in general and its sub-disciplines have evolved along recent years following basically two divergent lines: i) increasing complexity of questions, and ii) narrowing of the search space where answers are likely to be found.

QA systems can be seen as a natural extension of Information Retrieval (IR) systems. In the former IR systems the user information needs are expressed through a query, usually consisting on a set of keywords. The query is then used for retrieving information from a data set (a collection of documents in TREC challenges, the whole Web, a domain restricted collection, a corporate textual database, etc.). The output of IR consists of a, sometimes ranked, set of documents extracted from the data set. If the query is well formulated, a good system should retrieve the best ranked documents that satisfy the user information needs. In QA systems, the query consists of a NL question and the output of the system is not a set of likely relevant documents but the exact answer to the query.

Probably the most popular QA system is Watson (Ferrucci et al., 2010). Watson beated the best human champions at the US TV show *Jeopardy*. It has being turned into a tool for medical diagnosis. According to the IBM statements, its ability to analyze huge amounts of data is better than human doctors', and its deployment through the cloud reduces health-care costs.

Conventional QA systems are usually structured in four modules:

- Question Processing
- Information Retrieval of relevant documents
- Information Retrieval of passages or fragments
- Answer Extraction

So, the language technologies involved have to cover these 4 tasks:

- Linguistic analysis of questions using general or specific parsers (and grammars). This includes the definition of an appropriate tagset for classifying the questions
- Information Retrieval engines (general or specific for the task)

- Information Extraction techniques for extracting the answer

Initially QA was limited to Factoid Questions (Factoid QA), where the questions consisted on asking for a fact (a person, a date, a time, etc.). In Table 2.1 we present some examples of questions from TREC-8.

Example
How much folic acid should an expectant mother get daily?
Who invented the paper clip?
What university was Woodrow Wilson president of?
Where is Rider College located?
Name a film in which Jude Law acted.
Where do lobsters like to live?
Who was Picasso?

Table 2.1: Examples of factual questions from TREC-8

Answer these questions was not specially challenging because an assertive formulation of the answer is likely to be found in the collection. More complex questions should be answered, including temporal and spacial constraints. Consider the following example: *Who was the second USA republican president after the Vietnam war?*. To answer this question a QA system should probably split the complex question into a set of related simpler questions: *When did the Vietnam war ends?*, giving Date_1, *Give me the USA presidents after Date_1*, giving Person_1, Person_2, Person_3, ..., *To which party does Person_1 belong?*, and so on. Sometimes the complexity of the question arises from looking not for a fact but for a list of facts. Definitional QA systems are those looking for the definition of a person, an organization or simply a term. In these systems the answer probably has to be synthesized from partial pieces of information extracted from several documents.

Another important area is the domain-restricted QA, DRQA. Here both questions and search space are restricted to a given domain. Many domains have been faced: geography, tourism, economics, etc. Perhaps the domain object of the most applications is the medical domain.

Usually DRQA are applied to specific tasks and use domain specific lexicons, terminologies, knowledge bases, ontologies and other domain restricted lexico-conceptual resources. Search spaces are smaller and therefore, approaches based on the redundancy of answers (as voting techniques) are useless. User's requirements are usually high and system performance is more *precision*-oriented than *recall*-oriented (no answer is better than a wrong answer). Questions and documents are usually challenging and frequently contain acronyms, non-textual content (tables, itemized lists, etc.), domain specific jargon, etc. Two interesting systems in this scenario are QALM (Hallili et al., 2014), and SynchroBot (Cabrio et al., 2015).

Another active variant of QA is Community QA (CQA). In this scenario a member of the community formulates an initial query (a NL question) that triggers a thread of interventions of the community members that answer, refine and comment previous interventions forming a usually complex tangled thread. Members's interventions are usually related pairs of questions and answers. CQA have been recently evaluated in the framework of SEMEVAL-2015[1] to SEMEVAL-2017[2] contests. A review of the later contest can be found in (Rosenthal et al., 2017). The approach assumes that questions and answers share some common latent topics and are generated in a "question language" and "answer language", respectively, following the topics.

Recently, with the development of Linked Open Data, QA over linked data (QALD) has emerged as a popular discipline. QALD is a discipline placed in the intersection of QA and the Semantic Web. While more and more structured data is published on the Web, the question of how typical Web users can access this body of knowledge becomes of crucial importance. Over the last years, there has been a growing amount of research on interaction paradigms that allow end users to profit from the expressive power of Semantic Web standards, while at the same time hiding their complexity. Keyword queries, like common Web search nowadays, constitute a highly ambiguous and very impoverished representation of an information need. In the framework of the Semantic Web there has been recently a huge growth of available open and closed domain resources. Many of these resources are included into the LOD initiative. The most known and used resources in LOD are FreeBase[3], YAGO[4], YAGO2[5], and, specially, the DBPedia[6] as open domain LOD and BioPortal[7] for the medical and genomic domains. The QALD challenges, see (Unger et al., 2015), has contributed a lot to the development of these systems. (Kalaivani and Duraiswamy, 2012) survey different types of QA systems based on ontology and Semantic Web (SW) models with different query formats.

Focusing on the medical domain, (Dina Demner-Fushman, 2009) classify the type of questions occurring in clinical situations:

- Information on particular patients
- Data on health and sickness within the local population
- Medical knowledge
- Local information on doctors available for referral
- Information on local social influences and expectation
- Information on scientific, political, legal, social, management, and ethical changes affecting both, how medicine is practiced and how doctors interact with individual patients

Traditional approaches to question answering problems were based on rule-based

[1] `http://alt.qcri.org/semeval2015/`

[2] `http://alt.qcri.org/semeval2017/`

[3] http://freebase.com

[4] https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/

[5] http://resources.mpi-inf.mpg.de/d5/yago1$_{y}ago2$/

[6] http://dbpedia.org

[7] https://bioportal.bioontology.org/

models or linear classifiers over hand-engineered features (Ric, 2013),(Chen et al., 2016), or structured queries and the use of kowledge bases (Berant et al., 2013). However, the neural models have achieved great success in Natural Language Processing (NLP) tasks, usually without too much hyper-parameter tuning. Among the currents of implementation, three of them stand out, Convolution Neural Networks (CNN), due to their ability of capturing local correlations; the Recurrent Neural Networks (RNN), for their ability to capture long-term dependencies, specifically the Long Short Term Memory networks (LSTM) that avoid the gradient vanishing problem (Zhou et al., 2015a), and the Co-attention Networks, that allow to capture the interactions between the question and the document (Xiong et al., 2016). Many times, combinations of two ore more of these types of networks are used. We will focus on these approaches on next section.

2.2 Neural Models applied to Question Answering

In this section we are going to focus in neural models and explain some of the most interesting and recent implementations. First, we will discuss the work of (Kim, 2014), that used CNNs to classify questions, with a similar approach to the work of (Kalchbrenner et al., 2014). Next, we present (Zhou et al., 2015a) who used a combined architecture of CNN and LSTM for sentiment classification and question classification tasks. Then, we will describe attention-based models like the one in (Chen et al., 2017) and (Xiong et al., 2016), which is based on the work of (Lu et al., 2016). Finally, we will talk about the work of (Yih et al., 2013) based on word alignments for answer sentence selection.

Other interesting approaches not discused here are the work of (Bahdanau et al., 2014) who applied attention mechanism and Recurrent Neural Networks to learn alignments. Although (Bahdanau et al., 2014) focus on Machine Tranlation, their model can be applied to whatever mapping-based task as Summarization, Paraphrase detection, NL Inference, or CQA. (Qiu and Huang, 2015) used convolutional neural tensor network (CNTN) to retrieve similar questions, encoding the sentences in a semantic space and modeling their interactions with a tensor layer. (Chen et al., 2016) proposed a competitive statistical baseline using crafted lexical, syntactic, and word order features. (Yu et al., 2016) used a neural machine comprehension model that extracts answer candidates of variable lengths from the document and ranks them to answer the question.

(Kim, 2014) used a simple CNN architecture of a single layer and filters of different size (3, 4, 5) shown in Figure 2.1. They made experiments with one and two channels, both initialized with pre-trained word embeddings, trained with the Word2Vec model on Google News (Mikolov et al., 2013b), but allowing one of the channel's embeddings being trainable with backpropagation.

They showed that this simple architecture produces remarkable results with simple feature engineering and that the pre-trained word vectors perform good in different benchmarks. However, learning task-specific embeddings through fine-tuning

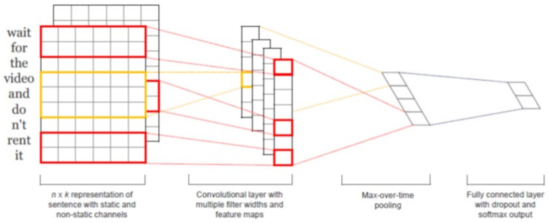

wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 2.1: CNN architecture of (Kim, 2014) model taken from (Kim, 2014).

leads to a even better performance. The words that were not contained in the pre-trained vectors were initialized randomly.

In the two channels configuration, the filter is applied to both channels and then added to produce a single feature map, which in essence, behaves like single channel network. Regarding to the feature maps, the optimal number was set to 100. Additionally, they used a drop-out regularization layer at 50%.

Among their observations they pointed out that the use of two channels had mixed results for different tasks and therefore, they kept the simpler version of using a single channel. Another important observation was the comparison with the results obtained by (Kalchbrenner et al., 2014), who used basically the same structure but with lower results; the authors attribute this difference to the use of different size of filters and the right number of feature maps.

(Zhou et al., 2015a) model sentences combining a CNN and a LSTM to benefit from the ability of the CNNs to learn local and global (using pooling) correlations and the ability of LSTM of learning sequential correlations.

They built a combined architecture called C-LSTM where the feature maps of a 1-layer CNN are arranged as sequential window features (n-gram) to feed a LSTM. The input of the CNN are Word2Vec pre-trained word vectors, trained with Google News Dataset. Those words not present in the pre-trained vectors, were initialized with a uniform distribution.

The output of the hidden layer at the last time step of the LSTM is the final sentence representation over which a softmax layer is put on top to make the class prediction. They used cross-entropy as the loss function, Stochastic Gradient Descent (SGD) to learn the parameters and RM-Sprop as optimizer.

This combined architecture outperformed all published neural baseline models at that time. Is worth to mention that one of their initial assumptions was that several parallel convolutional layers with different filter sizes would lead to a better performance, however, they discovered that a single convolutional layer with a filter of length 3 was the best option.

(Xiong et al., 2016) address the task of finding answers given a set of documents. In their approach they try to predict the *start* and *end* tokens of the span that contains the answer. They introduced the Dynamic Coattention Networks (DCN), based on the work of (Lu et al., 2016) for visual question answering. The

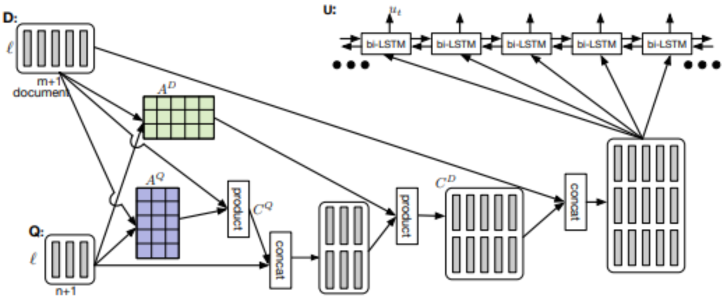


Figure 2.2: Coattention encoder of (Xiong et al., 2016) model taken from (Xiong et al., 2016)

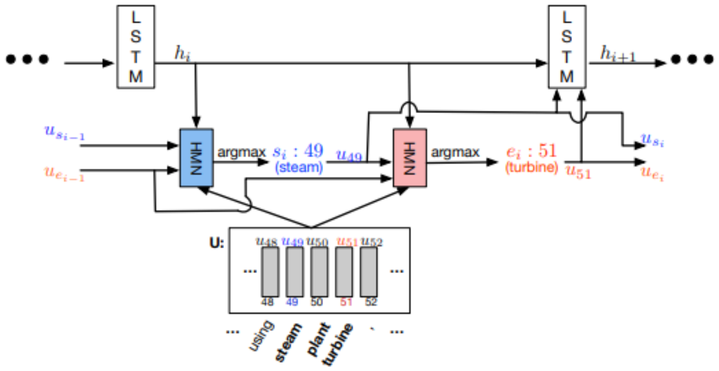


Figure 2.3: Dynamic decoder of (Xiong et al., 2016) model taken from (Xiong et al., 2016).

DCN consists on two principal components: a *coattention encoder*, shown in Figure 2.2, that captures the interactions between the question and the paragraph, and a *dynamic pointing decoder*, shown in Figure 2.3, that iterates over potential answer spans. These networks aim to solve the problem of other deep learning models regarding to getting stuck in local maxima, corresponding to wrong answers. Additionally, their results showed that this type of networks is not affected by the length of the documents.

The representation of the question Q and the paragraph that likely contains the answer P are obtained using an LSTM on top of word embedding vectors (Emb_{p_i}), where $\{p_1, \ldots, p_m\}$ are the words of the paragraph, producing the encoded vectors enc_{p_i}:

$$enc_{p_i} = LSTM(enc_{p_{i-1}}, Emb_{p_i}) \tag{2.1}$$

The same process is applied for the question embeddings Emb_{q_i}, where $\{q_1, \ldots, q_n\}$ are the words in Q:

$$enc_{q_i} = LSTM(enc_{q_{i-1}}, Emb_{q_i}) \tag{2.2}$$

In the **coattention encoder** M_P is defined as the paragraph encoding matrix $[enc_{p_1} \dots enc_{p_m} enc_{p\emptyset}] \in \mathbb{R}^{l \times (m+1)}$, where l is the length of the vector and every vector enc_{p_i} is concatenated column-wise. M_Q is defined as the question encoding matrix $[enc_{q_1} \dots enc_{q_n} enc_{q\emptyset}]$. Both matrices have a sentinel vector ($enc_{p\emptyset}$ and $enc_{q\emptyset}$) to not attend to any particular word in the input.

The final representation of the question Q (enc_Q) is obtained adding a non-linear projection layer (tanh) over of the question encodings (M_Q). For more details, see (Xiong et al., 2016).

The coattention mechanism consists on calculating an affinity matrix L with all the pairs of question words and paragraph words ($L = M_P enc_Q \in \mathbb{R}^{(m+1)\times(n+1)}$). Then a softmax function is aplied row-wise to produce the attention weights A_Q across the paragraph for each word in the question and column-wise to obtain the attention weights A_P for every word in the paragraph:

$$A_Q = softmax(L) \in \mathbb{R}^{(m+1)\times(n+1)} \qquad A_P = softmax(L^T) \in \mathbb{R}^{(n+1)\times(m+1)} \tag{2.3}$$

Next, attention contexts (C_Q) of the paragraph considering each word in the question are computed as follows:

$$C_Q = M_P A_Q \in \mathbb{R}^{l \times (n+1)} \tag{2.4}$$

The final *coattention context* C_P is a co-dependent representation of the question and paragraph. Is calculated using the context of the question considering each word in the paragraph ($M_Q A_P$) computed jointly with the mapping of the question context over the paragraph encodings space ($C_Q A_P$), concatenating M_Q and C_Q horizontally:

$$C_P = [M_Q; C_Q] A_P \in \mathbb{R}^{2l \times (m+1)} \tag{2.5}$$

The final step is to incorporate temporal information to the coattention contex through a bidirectional LSTM ($Bi - LSTM$):

$$u_i = Bi - LSTM(u_{i-1}, u_{i+1}, [enc_{p_i;C_{D_i}}]) \in \mathbb{R}^{2l} \tag{2.6}$$

The resulting matrix $U = [u_1, \dots, u_m] \in \mathbb{R}^{2l \times m}$ is the coattention encoding that helps to determine which span may be the best possible answer.

The other component of the dynamic coattention network is the **dynamic pointing decoder**. This is an iterative mechanism that alternates between predicting the

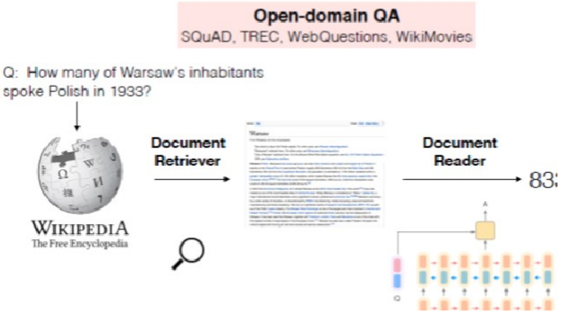

Open-domain QA
SQuAD, TREC, WebQuestions, WikiMovies
Q: How many of Warsaw's inhabitants spoke Polish in 1933?
WIKIPEDIA
The Free Encyclopedia
Document Retriever
Document Reader
833

Figure 2.4: Architecture of (Chen et al., 2017) model taken from (Chen et al., 2017) .

start and the *end* token of the span, so the algorithm does not get stuck in answer candidates located in different paragraphs when a document is provided as input.

In every iteration i the decoder updates its state using the information from the coattention encodings (U) of the current candidates of *start* (s_i) and *end* (e_i) tokens. Then, estimates of the new candidates are computed using a multi-layer LSTM. Denoting with h_i the hidden state of the network, the update is defined as:

$$h_i = LSTM(h_{i-1}, [u_{s_{i-1}}; u_{e_{i-1}}]) \tag{2.7}$$

where $u_{s_{i-1}}$ and $u_{e_{i-1}}$ are the coattention encodings of the *start* and *end* candidates of the previous iteration. The current *start* and *end* tokens would be the words that maximize the start score (α_i) and end score (β_i), respectively.

To calculate the start score α_i and end score β_i the authors proposed a combination of the Maxout Network of (Goodfellow et al., 2013) and the Highway Network of (Srivastava et al., 2015) called Highway Maxout Network (HMN), the implementation details can be found in (Xiong et al., 2016).

Finally, the network is trained minimizing the cummulative softmax cross-entropy of the *start* and *end* candidates in each iteration.

(Chen et al., 2017) also tackle the problem of finding answers inside paragraphs using attention scores. The overall architecture is presented in Figure 2.4. They built feature vectors for the questions and paragraphs inside the documents using Recurrent Neural Networks. These vectors are later used as input of two bilinear classifiers that independently predict the *start* and *end* tokens of the span of tokens containing the right answer.

They built an Open-Domain question answering system using Wikipedia as only source of answers. As the first step of the system, they created a document retrieval system that uses inverted index lookup to narrow the search, followed by a TF-IDF weighted bag-of-words vector model to obtain the five most relevant articles.

Having narrowed the search, they used what they called the *Document Reader* system to extract answers from a single paragraph or a small collection of paragraphs. This Document Reader consists on a recurrent neural network model that encodes feature vectors, that would be later used as input of two bilinear classifiers to predict the span of tokens that contains the answer.

A single paragraph P consists on a set of m words $\{p_1, \ldots, p_m\}$ and a question Q consists on a set of n words $\{q_1, \ldots, q_n\}$. The paragraph words are first transformed into feature vectors $feat_{p_i}$ composed by four different parts: a GloVe (Pennington et al., 2014) pre-trained word embedding vector Emb_{p_i}. Second, a binary disjoint set that indicates if p_i matches a token (q_j) of question Q, represented as $\amalg(p_i \in Q)$. Third, a triplet formed by syntactic inherent features of the word, specifically, its part-of-speech (POS) and named entity recognition (NER) tags and its (normalized) term frequency (TF), resulting in a triplet of the form $(POS(p_i), NER(p_i), TF(p_i))$. As last, an aligned question embedding defined as $Emb_{aligned}(p_i) = \sum_j a_{i,j} Emb_{q_j}$, where $a_{i,j}$ is the attention score that captures the similarity between the paragraph token p_i and each token q_j of the question Q. $a_{i,j}$ is computed by the dot products between nonlinear mappings of word embeddings:

$$a_{i,j} = \frac{exp(\alpha(Emb(p_i)) \cdot \alpha(Emb(q_j)))}{\sum_{j'} exp(\alpha(Emb(p_i)) \cdot \alpha(Emb(q'_j)))} \tag{2.8}$$

and $\alpha(\cdot)$ is a single dense layer with ReLU nonlinearity. These aligned question embeddings provide a soft alignment between similar but not identical words.

These feature vectors $feat_{p_i}$ are then encoded by a multi-layer bidirectional long short-term memory network, to extract context information. The process is illustrated as:

$$enc_{p_i} = Bi - LSTM(enc_{p_{i-1}}, enc_{p_{i+1}}, feat_{p_i}) \tag{2.9}$$

where the enc_{p_i} is the concatenation of each layer's hidden units.

In a similar way, the question feature vectors $feat_{q_i}$ are built using the word embedding vectors of the question tokens (Emb_{q_i}) as input of another recurrent neural network, where the resulting hidden units are combined into one single vector enc_Q:

$$enc_{q_i} = Bi - LSTM(enc_{q_{i-1}}, enc_{q_{i+1}}, feat_{q_i}) \tag{2.10}$$

$$enc_Q = \sum_j b_j(enc_{q_j}) \tag{2.11}$$

where b_j encodes the importance of each question word and w is the weights vector to learn:

$$b_j = \frac{exp(w \cdot enc_{qj})}{\sum_{j'} exp(w \cdot enc_{qj'})} \tag{2.12}$$

Finally, they used a bilinear term to predict the *start* and *end* tokens of the span of tokens that contains the answer, using the similarity between enc_{pi} and enc_Q and computing the probability of each token p_i being the *start* ($Prob_{start}(p_i)$) or *end* ($Prob_{end}(p_i)$) token. The classifier chooses the span from the token p_i to token $p_{i'}$ that maximizes $Prob_{start}(p_i) \times Prob_{end}(p_{i'})$.

Since some of the training data sets only contained the pairs of question and answer, without an associated document or paragraph, they could not use them to train the Document Reader. Therefore, they used *Distantly Supervised Data*, based on the work of (Mintz et al., 2009). The method consists on using question-answer pairs to look for the 5 most relevant Wikipedia articles and filtering the paragraphs that did not have an exact match with answer, the ones shorter than 25 and larger than 1500 characters and those that did not contain all the named entities detected in the question.

In another work, **(Yih et al., 2013)** built an answer selection system, where each question is associated to a set of labeled candidate sentence answers. Their approached was to detect a semantic match between the question and the answer. To solve this problem, they used *word-alignments* following the (Chen et al., 2017) structure. As a constraint, each word in the question needs to be linked to a word in the sentence and each word in the sentence can be linked to zero or multiple words in the question.

The idea was to use the underlying relations between the words of the question and the words of the answer with the aim of finding an association between both entities. The association strength could be measure in terms of degree of mappings between the words and the appearance of all question tokens in the answer. To reveal this underlying relation they used semantic components from lexical semantic models, including synonymy/antonymy, hypernymy/hyponymy (the Is-A relation) and general semantic word similarity.

They used two approaches to build a model capable of detecting the semantic relation between question and answer: bag-of-words and latent structures. In the first one, every pair (q_i, s_k), where q_i is a word in question Q and s_k is a word in a candidate answer sentence S, is transformed into a *d-dimensional* real-valued feature vector by feature functions $\phi_1, \ldots, \phi_d$. Then, this feature vectors are aggregated using *average* and *max* to obtain the whole question-answer pair feature vector (Φ):

$$\Phi_{avg_j} = \frac{1}{mn} \sum_{i \in |Q|, k \in |S|} \phi_j(q_i, s_k)$$

$$\Phi_{max_j} = max \sum_{i \in |Q|, k \in |S|} \phi_j(q_i, s_k)$$

producing a *2d-dimensional* feature vector for every question-answer pair. These vectors are then fed to two kinds of binary classifiers, a Logistic Regression model and a boosted decision tree.

On the other hand, to learn the latent structures they used Learning Constrained Latent Representations (LCLR), whose idea is to replace the decision function of standard linear models ($W^T\phi(x)$) with:

$$argmax_h W^T \phi(x, h)$$

where W represents the *weights* vector, x is the pair question-sentence (Q, S) and h is the word-alignment between Q and S. The final objective function is defined as:

$$\begin{aligned} \min_W \quad & \frac{1}{2} \parallel W \parallel^2 + C \sum_i \xi_i^2 \\ \text{s.t.} \quad & \xi_i \geq 1 - y_i \underset{h}{max} W^T \phi(x, h) \end{aligned}$$

where where $y_i = 1$ indicates that S is a correct answer to question Q_i.

In addition to the previously described features obtained from the properties of the pairs of words of the question and the answer, the authors added rich lexical semantic information that they split in 6 different categories: identical word matching (I), lemma matching (L), WordNet (WN), enhanced Lexical Semantics (LS), Named Entity matching (NE) and Answer type checking (Ans). All the features, except (Ans), were weighted by the IDF value of the question word. These nre features helped to improve the model performance.

3 | Background

3.1 Convolutional Neural Networks (CNN)

Because our architecture is based on convolutional neural networks, we are going to explain the main characteristics and the overall process of a CNN. The explanation is done using images as input since it is easier to explain that way.

Convolution Neural Networks or CNNs are hierarchical neural networks whose *convolutional* layers are alternated by *downsampling* layers. Since *Alex Krizhevsky et al. (2012)* used them in the ImageNet image classification challenge, obtaining better results than the state of the art at that time, CNNs have been greatly used in image classification. The CNNs are capable to distinguish low level features like curves and edges and build more complex patterns that allow to characterize an object. The most common architecture for classification is a series of convolutional layers where nonlinear functions are applied, pooling (downsampling) layers, and fully connected layers. (Ciresan et al., 2011).

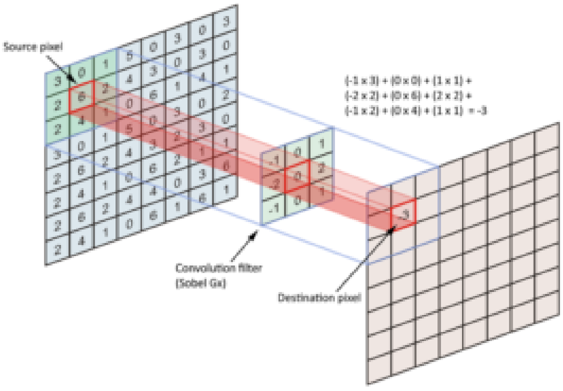

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel
-3

Figure 3.1: The process of convolution[1]

In image processing, the input of the CNNs are images represented as a grid of pixels with numeric values that indicate the intensity of the color in that spot. For color (RGB) images this grid has three layers or *channels*, where each one of those contain the intensity values for a different color (Red, Green or Blue).

Having the image grid, the convolutional layer works as a flashlight that focuses in a portion of the grid at a time and moves (*convolutes*) through the whole space (Deshpande, 2016). This flashlight is called the *filter* or *kernel*, can have a squared

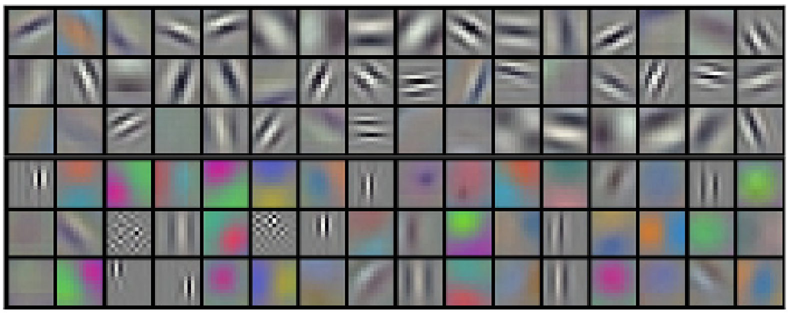


Figure 3.2: Features Map from 2 dimensional CNN for image classification (Krizhevsky et al., 2012).

or rectangular shape, e.g 5 × 5 pixels, and must have the same depth as the input, i.e if it is a RGB image the depth is 3 (5 × 5 × 3). This window will move through the whole image in the x and y axes by a specific number of steps. This steps are called *strides*, so if the the strides are equal to $(2, 2)$, the window will move two pixels horizontally and two pixels in the vertical direction. Figure 3.1 depicts the action of convolution in a 2 dimensional setting.

The filter also contains numbers that represent the *weights* or *parameters* that have to be estimated by the model using *backpropagation*. This weights are element-wise multiplied with the original values in the pixels, producing several multiplications that are then summed up, producing a single number. This process is repeated over the whole grid, obtaining a single number for every time the filter moves over the grid. All this resulting numbers will end up forming a lower dimensional matrix called *feature map* see Figure 3.1, and it is called that way because it extracts the *features* or low-level characteristic of the image, like textures, edges or straight lines (Deshpande, 2016). Figure 3.2 presents graphically the resulting feature maps for a task of image classification. The figure shows that lines with different orientations have been recognized. The dimensions of the output feature map (M) are:

$$M_x^n = \left\lfloor \frac{M_x^{n-1} - F_x^n}{S_x^n} \right\rfloor + 1; M_y^n = \left\lfloor \frac{M_y^{n-1} - F_y^n}{S_y^n} \right\rfloor + 1$$

Where M_i^n is the feature map of the layer n on the i axis, F_x^n and F_y^n are the dimensions of the filter in x and y axes and S_x^n and S_y^n are the strides in the horizontal and vertical directions (Ciresan et al., 2011).

The idea is to have several filters that detect different features, which produces an stack of feature maps of the same size. The feature maps will contain higher values (activate) in the positions corresponding to the regions of the image where the filter have detected the feature. For example, a cell in a map produced by a filter that identifies diagonal lines will have a high value if in its corresponding image region there was a diagonal line.

In Natural Language Processing, numeric representations of the words and sentences are used instead of pixels, and other considerations must be taken, for instance the frequent use of 1 dimension CNN instead of the usual 2 dimension CNN, as we

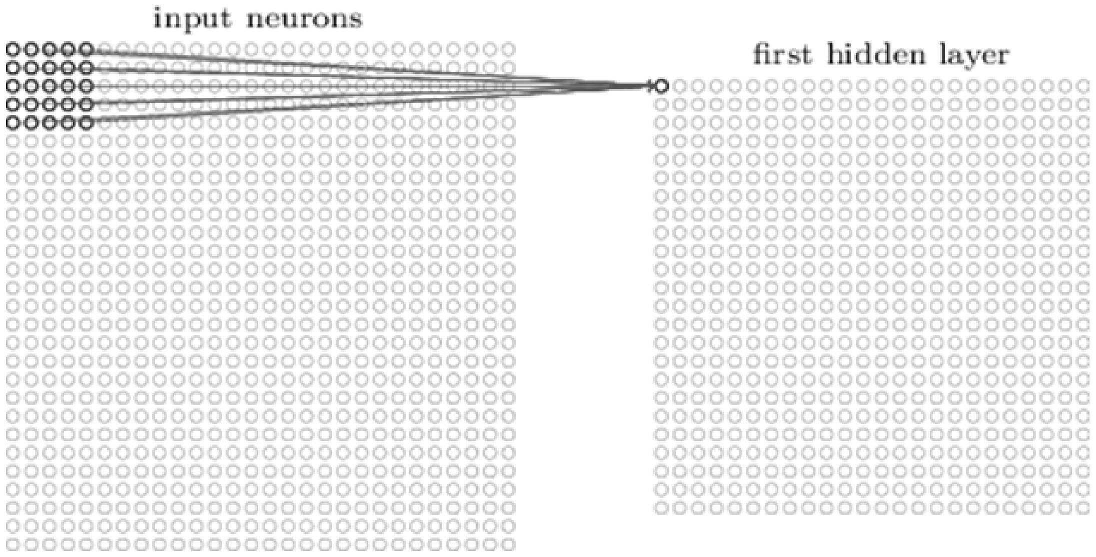


Figure 3.3: Example of a 5 × 5 filter producing an entry in the feature map (Deshpande, 2016)

described in the *State of the Art* chapter, but the overall process remains basically the same.

3.1.1 Pooling layer

When using CNNs is common the use of pooling layers that aggregate the joint effect of a neighborhood of features with the aim of producing invariance to small changes in the input. This produces robustness in terms of generalization and helps the convergence, preserving the relevant information and discarding non important details (Boureau et al., 2010). The most common aggregation function is the maximum (Max Pooling layer). Also summation, average, and k-max pooling are used.

3.2 Gradient Descent

Because the backpropagation training process of the CNN consists on determining iteratively the optimal weights or parameters in the *filter*, an optimization algorithm is required. Gradient descent is one the most popular algorithms to perform optimization and is widely used to optimize neural networks. There are different variants of gradient descent, including powerful methods like Newton's method, but that end up being impractical because of the computational cost of calculating the inverse of Hessian matrix of second derivatives. On the other hand, the most popular approaches only need first order information, calculating the gradient of the objective function with respect to the parameters. Among these simpler variants is the classical Stochastic Gradient Descent, that have a fast convergence rate. Adagrad is also one of these algorithms that only require first order information and it adapts the learning rate performing larger updates for infrequent parameters and smaller updates for frequent ones. Adadelta and RMSprop, are extensions of Adagrad and try to reduce the aggressive reducing learning rate, adapting it over each iteration and slowing down when approaching a local minima. Adaptive Moment Estimation (Adam) method is similar to RMSprop but adds bias-correction and momentum (Ruder, 2016).

3.3 Word2Vec Embeddings

Most of the neural models used for NLP use embeddings as word and sentence representations due to the great performance shown in several NLP tasks (Baroni et al., 2014). Neural models incorporate these vectors adding an embedding layer, sometimes static and sometimes dynamic.

Embeddings are dense vector representation of words or sentences in a lower dimensional space that contain many linguistic regularities and patterns. (Turian Joseph, 2010).

The most popular domain free embeddings are Word2Vec and GloVe. Recently Sampo Pyysalo has delivered a embedding data set for the medical domain learned from the whole collection of Medline articles. Details on these embeddings are presented in Section 4.2.

Several efforts have been made recently for combining embeddings of different characteristics and mapping embeddings between different languages. For instance, for medical texts Pyysalo's and Word2Vec or GloVe are somehow complementary and a combination seems to be beneficial. The task is not easy because the vector spaces are different. (Mikolov et al., 2013b) signaled that continuous word embedding spaces exhibit similar structures across languages and performed mappings between embeddings. (Mikel Artetxe and Agirre, 2016) and (Samuel L Smith and Hammerl, 2017) improved these cross-lingual word embeddings using all of them bilingual word vocabularies. A notable achievement is the system MUSE developed by Facebook group (Conneau et al., 2017) that using FastText obtained a high number of embeddings that cover more than one hundred languages.

Although embeddings of words has gotten excellent results in many NLP tasks this is not the case for embeddings of more complex entities such as phrases, sentences, rdf triples, and others. Initial attempts for embedding such entities usually consisted on using BOW for representing these entities and use summation, average, or dot product of the vectors for representing the embeddings. Recently, however, more semantically-based approaches have been followed for facing the task.

(Conneau and Kiela, 2018) have made public SentEval, an evaluation toolkit for evaluating sentence embeddings against several NLP task.

3.3.1 Word2Vec

Word2Vec is an algorithm created by (Mikolov et al., 2013a) that receives a corpus as input, builds the vocabulary and returns the vector representation of every word. Word2Vec has a simple architecture and a low complexity, which make it ideal to train high-dimensional vectors from big data sets. The algorithm provides support for two representations, bag of words and skip-grams. The bag of words architecture predicts the current word based on the context, and the skip-gram predicts surrounding words given the current word. (Mikolov et al., 2013a) tested both architectures and found out that the skip-gram model performs slightly worse than BOW in syntactic task, but much better the semantic part. Because of this, we chose to use Word2Vec with skip-grams.

3.3.2 Skip gram model

The skip-gram model is an efficient method for learning high quality vector representations of words where, given a word w, it predicts the surrounding words inside an specific perimeter or window using a log-linear classifier. While increasing the window, the quality of the resulting word vectors increases, as it does the complexity. Because more distant words have less impact, they are given less weight (Mikolov et al., 2013a). The resulting vectors are able to express similiarities between words using simple algebra operations. For example, vec("Madrid") - vec("Spain") $+$ vec("France") is closer to vec("Paris") than any other word, vec(w) being the vector representation of the word w.

4 | Thesis Statement and Implementation

In this chapter we are going to describe the work done in this thesis, the methodologies used and the contributions. First, in the Corpus section (4.1) we will introduce the corpus we worked with, the alternative sources we added and the two mechanisms that we implemented to increase the number of samples. Then in the Question Representation section (4.2), we will talk about the two encoding techniques to transform the questions into structured input for the models, including the Bag-of-words and embedding vectors. In the last section (4.3) about the Architecture, we will describe the classification models, including three different convolution neural network implementations and the use of a simple Logistic Regression. To classify the questions, we performed a Multiclass and a Multiclass-Multilabel classifications, taking into account four (general) and thirteen (general + specific) classes, respectively.

4.1 Corpus

In this section we will describe the corpus we used to train the classifier, including the two levels of labels we have. The first group of labels corresponds to the classes that describe the type of expected answer. The second group describes the topic of the questions. We are also going to explain the techniques we used to increase the number of available questions.

The corpus consists on 2,750 questions obtained from BioASQ[1] question answering challenge, with a vocabulary size of 5,440 words. All the questions are from the biology and medical domains and are grouped into four categories: 780 are *factoid*, 746 *yesno*, 669 *summary* and 556 *list*, with sentences of maximum of 33 words. These classes were manually annotated by PubMed curators and constitute a fairly balanced data set. The description of each category is in the Table 4.1.

Additionally, because we wanted to provide more information about the questions, we built our own categories based on their topic. This specific labels were added manually and are describe in the Table 4.1. While the general type of questions refer to the type of expected answer, these specific categories refer to the topic of the question. The percentages in the table indicate the proportion of each

[1] `http://www.bioasq.org`

Type of Question	Description
Yes/no	These are questions that, strictly speaking, require "yes" or "no" answers, though of course in practice longer answers will often be desirable. For example, "Do CpG islands colocalise with transcription start sites?" is a yes/no question.
Factoid	These are questions that, strictly speaking, require a particular entity name (e.g., of a disease, drug, or gene), a number, or a similar short expression as an answer, though again a longer answer may be desirable in practice. For example, "Which virus is best known as the cause of infectious mononucleosis?" is a factoid question.
List	These are questions that, strictly speaking, require a list of entity names (e.g., a list of gene names), numbers, or similar short expressions as an answer; again, in practice additional information may be desirable. For example, "Which are the Raf kinase inhibitors?" is a list question.
Summary	These are questions that do not belong in any of the previous categories and can only be answered by producing a short text summarizing the most prominent relevant information. For example, "What is the treatment of infectious mononucleosis?" is a summary question.

Table 4.1: General question categories

question type that is found in the data set; as we can see, the classes are pretty unbalanced.

4.1.1 Increasing the number of samples

Because the classification algorithms and specially NNs perform better with a high number of samples, we decided to use an alternative source of questions. The eligible data sets we considered to use are presented in Table 4.1.1. From them, we decided to choose Quora data set because it has a high number of questions and they are grouped by pairs, which would help us with the classification. Quora[2] is a question-and-answer site for asking and answering questions, that are organized by its community of users in the form of opinions. This data set contains 404,302 unlabeled pairs of questions that are potential duplicates and have the same answer. An example of the type of pairs of questions found in the data set are *How can I increase the speed of my internet connection while using a VPN?* and *How can Internet speed be increased by hacking through DNS?*. To take profit of these unlabeled questions we implemented two different approaches described below.

[2] https://www.quora.com/

Type of Question	Description
Condition (7%)	These are questions that refer to general concepts about medical conditions, diseases or syndromes (a condition is not necessarily a disease). For example, "What is the risk of developing acute myelogenous leukemia in Fanconi anemia?"
Condition detection (2.4%)	These are questions that talk about the methods and techniques to detect or identify a condition or disease. For example, "How can the fetal Rhesus be determined with non-invasive testing?"
Condition prevention (1.7%)	These are questions that talk about the methods to prevent a condition or disease. For example, "List clinical trials for prevention of sarcopenia".
Condition symptoms (4.5%)	These are questions that talk about the symptoms or characteristics of a condition or disease. For example, "List symptoms of Hakim Triad".
Condition treatment (11.7%)	These are questions that talk about the treatment of a condition or disease. For example, "Is Bladder training an effective method to treat urge incontinence?".
Condition cause (17.6%)	These are questions that talk about the at least one of the causes, natural or not, of a condition or disease. For example, "Which fusion protein is involved in the development of Ewing sarcoma?" (Li et al., 2010).
Drug (5.4%)	These are questions that talk about the general characteristics of a drug. For example, "What are the main indications of lacosamide?".
Biology (42.3%)	These are questions that talk about general concepts in biology. For example, "What is the structural fold of bromodomain proteins?".
Medicine (7.4%)	These are questions that talk about methods, studies of techniques related to medicine. For example, "How many clinical trials for off-label drugs in neonates are cited in the literature".

Table 4.2: Specific question categories

Co-training approach

Taking inspiration from (Chen et al., 2017) about using distantly supervised data, to label the extra questions we used our already trained and tuned CNN model. If the same class was predicted for both questions of the pair, they were assigned to that class. On the contrary, if the questions were assigned different labels, they were discarded.

Data set	N. of questions	Description
quasarT	43000	Open-domain trivia questions.
quasarS	37000	Fill-in-the-gap queries constructed from definitions of software entity tags on Stack Overflow.
SQuAD1.1	98169	Stanford Question Answering Dataset (SQuAD) is a reading comprehension data set, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable.
SQuAD2.0	142192	Combines the 100,000 questions in SQuAD1.1 with over 50,000 new, unanswerable questions written adversarially by crowdworkers to look similar to answerable ones. To do well on SQuAD2.0, systems must not only answer questions when possible, but also determine when no answer is supported by the paragraph and abstain from answering.
TREC (Curated)	2180	TREC questions with answers added.
WikiMovies	181679	This includes only the QA part of the Movie Dialog data set, but using three different settings of knowledge: using a traditional knowledge base (KB), using Wikipedia as the source of knowledge, or using IE (information extraction) over Wikipedia. This allows to test the ability of models to directly read documents to answer questions, and to compare this to traditional KBs in the same setting.
TriviaQA	109767	reading comprehension data set containing over 650K question-answer-evidence triples. TriviaQA includes 95K question-answer pairs authored by trivia enthusiasts and independently gathered evidence documents, six per question on average, that provide high quality distant supervision for answering the questions.
Quora	404302	Allows to train and test models of semantic equivalence, based on actual Quora data. The dataset consists of over 400,000 lines of potential question duplicate pairs. Each line contains IDs for each question in the pair, the full text for each question, and a binary value that indicates whether the line truly contains a duplicate pair.
WikiQA	29261	Set of question and sentence pairs, collected and annotated for research on open-domain question answering. In addition, the WikiQA data set also includes questions for which there are no correct sentences, enabling researchers to work on answer triggering, a critical component in any QA system.

Table 4.3: QA data sets description

Question Patterns

Another method we applied to increase the number of samples was to try to find significant patterns based the structure of the questions using their *Part Of Speech* (POS) tags, for later, classify the Quora questions based on these patterns. In the Table 4.1.1 we describe the four versions of the POS tag patterns built, including an example for the question *What is a Caveolae?*.

Type of Patterns	
Extended	Part of speech tags of the words in the question. Example: (WP,VBZ,DT,NNP,.)
Reduced	Part of speech tags of the words in the question, but only keeping the first tag letter. Example: (W,V,D,N,.)
Reduced + stop words	Part of speech tags of the words in the question, but keeping only the first tag letter and the stop words. Example: (W,is,a,N,.)
Reduced + stop words + punctuation signs	Part of speech tags of the words in the question, but keeping only the first tag letter, the stop words and punctuation signs. Example: (W,is,a,N,?)

Table 4.4: POS tag patterns

To find the significant patterns we first clustered the BioASQ questions based on the euclidean distance of their respective embedding vectors, hoping that the occurrence of two questions in the same cluster meant they were similar. Luckily there will be a predominant class in the cluster. To perform the clustering we used *K-means* algorithm, setting the number of clusters to 8 using a visual representation of the hierarchical clustering.

An ideal pattern would be the one that occurs with a *high* frequency in only one cluster and whose associated questions had the same class. In this way, we would have a set of patterns highly correlated with a certain type of questions that latter would help us to classify unlabeled ones. In the Table 4.1.1 we present an example of the clusters obtained for the extended version.

Finally, when a new question is given, it would be classified to the class associated with the pattern that matches the question's pattern, if any. In this way we could add more samples to the corpus.

4.2 Question Representation

In order to extract information from unstructured data like text or images, a change of representation must be applied to transform it into structured data that can be consumed by the algorithms. We will call the input to the models the *input matrix.*

In this section we are going to present the methods we used to encode the ques-

Pattern	**Cluster**	**Frequency**	**Class**	**Questions**
VBZ,NNP,VBN,IN,NN,NN,NN,.	7	2	'yesno'	*Is Rac1 involved in cancer cell invasion?* *Is CHEK2 involved in cell cycle control?*
MD,VB,VB,JJ,JJ,NN,.	6	2	'yesno'	*Can acupuncture cause spinal epidural hematoma?* *Can canagliflozin cause euglycemic diabetic ketoacidosis?*
WP,NN,VBZ,VBN,IN,NNP,.	4	2	'factoid'	*What enzyme is inhibied by Opicapone?* *What molecule is targeted by Avelumab?*
NNP,VB,NN,JJ,NN,NN,.	4	2	'yesno'	*Does thyroid hormone affect cardiac remodeling?* *Does thyroid hormone affect cardiac remodeling ?*
WP,VBP,DT,JJ,NNS,IN,NNP,.	0	2	'list'	*What are the side effects of Nalmefene?* *What are the generic versions of Viagra?*
WP,VBZ,DT,JJ,NN,IN,DT,NNP,NN,.	4	2	'factoid'	*What is the molecular function of the Chd1 protein?* *What is the characteristic feature of the Dyke-Davidoff-Masson syndrome.*
VBZ,NNP,VBN,IN,JJ,NN,.	6	2	'yesno'	*Is Propofol used for short-term sedation?* *Is ABCE1 involved in ribosomal recycling?*
NNP,EX,DT,JJ,NNS,IN,NN,.	1	2	'yesno'	*Are there any specific antidotes for rivaroxaban?* *Are there any specific antidotes for dabigatran?*
NN,NNS,JJ,IN,NN,NN	4	2	'list'	*List programs suitable for pharmacophore modelling* *List programs suitable for protein docking*
NNP,VBZ,DT,NN,IN,WDT,NN,.	1	2	'factoid'	*Treprostinil is an analogue for which prostaglandin?* *Idarucizumab is an antidote of which drug?*
WDT,VBP,DT,JJ,NNS,IN,NN,NN,.	3	2	'list'	*Which are the main methods for pharmacophore modelling?* *Which are the cellular targets of imatinib mesylate?*

Table 4.5: Pattern Clusters

tions, starting with the classical sparse representation of Bag Of Words, followed but a dense representation using word and sentence embeddings. We built our own embedding vectors using the Word2Vec model (Mikolov et al., 2013b) and used alternative pre-trained vectors from GloVe (Pennington et al., 2014) and Pyysalo (Moen and Ananiadou, 2013).

4.2.1 Bag Of Words (BOW)

The first method we used for question representation was the classical *Bag Of Words* at sentence level, where every sentence behaves as a bag of its own words without taking into account the grammar or the order. This bags are n-hot encoding one-dimensional vectors of the size of the vocabulary, i.e. 5,540 in our experiments. In this approach every row (vector) of the input matrix represents a question, filling with 1's the positions of the words present in the question, and leaving the rest as 0's. This produces a sparse matrix of *number of questions* rows and *vocabulary size* columns.

Below there is an example of the vocabulary and the n-hot encoded vectors built for the the following set of two questions: {*Does BNP increase after intensive exercise in athletes?*, *What is the aim of the Human Chromosome-centric Proteome Project (C-HPP)?*}.

Vocabulary

Does	of	in	what	is	the	after	increase	intensive	exercise	athlete	?	aim

BNP	Human	Chromosome-centric	Proteome	(	C-HPP	)	Project

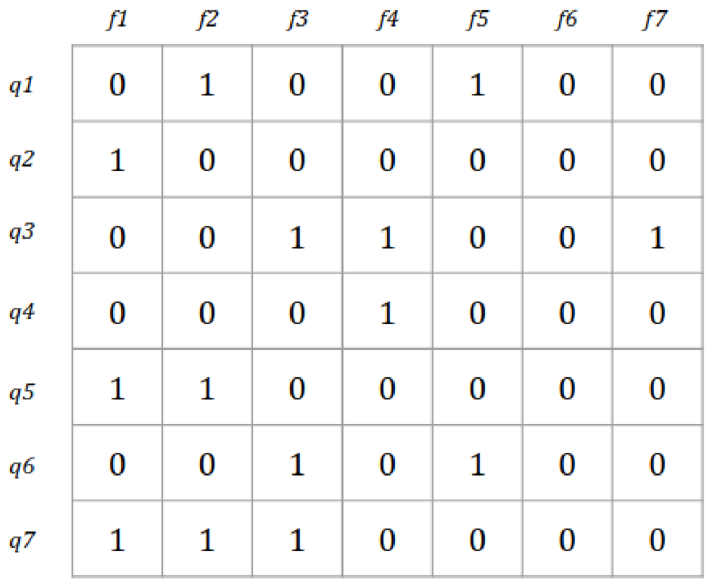
Does BNP increase after intensive exercise in athletes?

1	0	1	0	0	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0

What is the aim of the Human Chromosome-centric Proteome Project (C-HPP)?

0	1	0	1	1	1	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1

An example of a randomly generated n-hot encoded vectors matrix of seven questions and seven maximum number of tokens is presented in the Figure 4.1.



	f1	*f2*	*f3*	*f4*	*f5*	*f6*	*f7*
q1	0	1	0	0	1	0	0
q2	1	0	0	0	0	0	0
q3	0	0	1	1	0	0	1
q4	0	0	0	1	0	0	0
q5	1	1	0	0	0	0	0
q6	0	0	1	0	1	0	0
q7	1	1	1	0	0	0	0

Figure 4.1: Example of n-hot encoded vectors matrix, where each q_i corresponds to a question and each f_j corresponds the j_{th} word in the vocabulary.

The drawbacks of this approach is that the order of the words in the sentence is lost and it suffers from data sparsity, which produces for uncommon words poorly estimated parameters. Additionally, it could happen that some of the words in new data sets are not contained in the training vocabulary and the model could not process them. Because of this, we opted to use **Embeddings** at word and sentence level.

4.2.2 Embeddings

The aim of using embeddings is to detect linguistic regularities and patterns, so every embedding dimension would correspond to a *feature* that ideally contains a grammatical or semantic interpretation (Turian Joseph, 2010). We used two different kinds of embeddings: word-level and sentence-level, trained with the Word2Vec and Doc2Vec models, respectively (Mikolov et al., 2013b). We also used two additional sets of **pre-trained vectors**: the first ones trained with the GloVe (Global Vectors) model (Pennington et al., 2014), choosing the version that contains 8,40B tokens and 300-dimensional vectors. GloVe embeddings are trained on the non-zero entries of a global word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another in a given corpus (Pennington et al., 2014).

Oher pre-trained embeddings used were the Pyysalo's (Moen and Ananiadou, 2013), which are medical domain 400-dimensional word vectors, containing 5.5B tokens induced from PubMed and PMC. Pyysalo embeddings are also trained with Word2Vec using the skip-gram model with a window size of 5.

Is worth mentioning that we did a pre-filtering of the words in the pre-trained embeddings vectors, keeping only the words present in our vocabulary. However, there were cases when a word in the vocabulary was not found among the pre-trained vectors. For this reason, we used variations of the original words in the vocabulary to try to find a match. This word variations included lower and upper cases, plural and singular forms, lemmas and regular expressions that only matched letters and took out any symbol. With these variations we could increase the number of embeddings vectors associated to each word. However, if after comparing with the word variations there was not a match, the missing word was assigned to a 0-filled empty vector.

The process of building our **own-trained embeddings** consisted on identifying the syntactic structure of the sentences and build the **feature vectors** using information available of the words. We first tokenized the questions using the Stanford Core NLP library and extracted two main features: the lemmas and the Part Of Speech (POS) tags. The lemmas constitute the base form of the words, removing any inflectional ending or alteration (Manning and Schütze, 2008). The part of speech corresponds to the disambiguated category of the word that is assigned in accordance to its syntactic function or its role in the sentence, like *noun*, *pronoun*, *verb*, *adjective*, etc. Among the POS tags, there is one that corresponds to the named entities (NNP), which is defined as a real-world object, like people, locations or organizations, that can be denoted with a proper name. In our data set there are many biology and medical named entities. Is worth mentioning that we did not remove stop words since they were useful to identify the type of the question.

For the lemma extraction we used the NLTK WordNetLemmatizer (Bird et al., 2009), which is based in WordNet, a large lexical database of English. For the part of speech tags we used the Standford NLP POS tagger (Kristina Toutanova and Singer, 2003).

To try to provide more context to the biomedical named entities we added a descriptive categorization using the BioPortal API[3], which is a repository of biomedical ontologies, that allows to retrieve information about biomedical terms. From all the information available, we focused on the semantic type of the named entities. For example, the search of the term 'RNAs' returns 'Nucleic Acid, Nucleoside, or Nucleotide'.

Because not all of the sentences had the same number of words, we decided to truncate the ones that overpass the 18 tokens, since the higher frequency of sentences length was 15. On the contrary, if they were shorter, we padded them with an *empty* word.

Here we present an example of the feature extraction process of the question *Does BNP increase after intensive exercise in athletes?*:

Tokens

Does	BNP	increase	after	intensive	exercise	in	athletes	?

[3] http://data.bioontology.org/documentation

Lemmas

Does BNP increase after intensive exercise in athlete ?

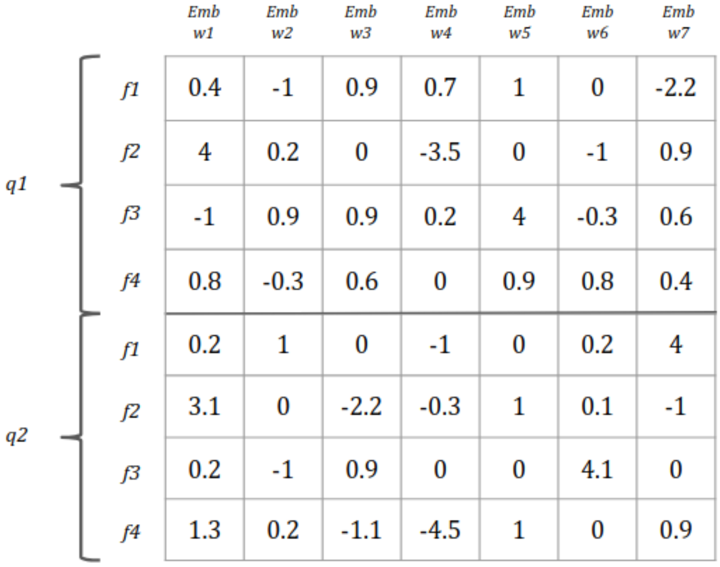
POS tags

VBZ NNP VB IN SS NN IN NNS '.'

After extracting the word features we built the **feature vector** of each question, which basically consists on the string concatenation of every word's lemma, part of speech tag and, if the latter corresponds to a named entity, its semantic type, all separated by slashes (/). Bellow we present the feature vector of the previous example.

'Does/VBZ' 'BNP/NNP/Amino Acid, Peptide or Protein' 'increase/VB' 'after/IN' 'intensive/JJ' 'exercise/NN' '?/.'

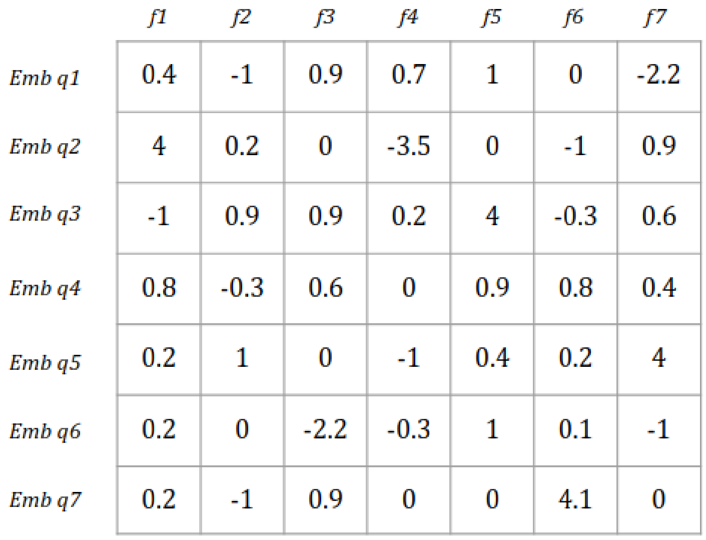
Finally, we built the embeddings ingesting the feature vectors into the **Word2Vec** model implemented in the Gensim Python library (Řehůřek and Sojka, 2010) using skip-gram and a window of 5 words. An example of a randomly generated word-level embedding vectors matrix is presented in the Figure 4.2.



		Emb w1	Emb w2	Emb w3	Emb w4	Emb w5	Emb w6	Emb w7
q1	f1	0.4	-1	0.9	0.7	1	0	-2.2
	f2	4	0.2	0	-3.5	0	-1	0.9
	f3	-1	0.9	0.9	0.2	4	-0.3	0.6
	f4	0.8	-0.3	0.6	0	0.9	0.8	0.4
q2	f1	0.2	1	0	-1	0	0.2	4
	f2	3.1	0	-2.2	-0.3	1	0.1	-1
	f3	0.2	-1	0.9	0	0	4.1	0
	f4	1.3	0.2	-1.1	-4.5	1	0	0.9

Figure 4.2: Example of **word-level** embedding vectors matrix, where each q_i corresponds to a question, each *Emb* w_j corresponds to the embedding vector of the j_{th} word of the question and every f_k corresponds to k_{th} dimension or feature of the embedding vectors.

In addition to the word-level embedding vectors, we also built **sentence-level** embedding vectors using the **Doc2Vec** model from Gensim Library (Řehůřek and Sojka, 2010), whose only difference with Word2Vec model is that it takes into account not only the word vectors but also other sentence embeddings, finally averaging the individual word vectors. An example of a sentence-level embedding vectors matrix is in the Figure 4.3.



	f1	*f2*	*f3*	*f4*	*f5*	*f6*	*f7*
Emb q1	0.4	-1	0.9	0.7	1	0	-2.2
Emb q2	4	0.2	0	-3.5	0	-1	0.9
Emb q3	-1	0.9	0.9	0.2	4	-0.3	0.6
Emb q4	0.8	-0.3	0.6	0	0.9	0.8	0.4
Emb q5	0.2	1	0	-1	0.4	0.2	4
Emb q6	0.2	0	-2.2	-0.3	1	0.1	-1
Emb q7	0.2	-1	0.9	0	0	4.1	0

Figure 4.3: Example of **sentence-level** embedding vectors matrix, where each *Emb* q_i corresponds to the embedding vector of the i_{th} question and every f_j corresponds to j_{th} dimension or feature of the embedding vectors.

4.2.3 Alternative feature vectors

Following our intuition that adding more information about the syntactic structure of the question would lead to a better classification performance, we built another version of the feature vectors.

Constituency tree

In this version we used the **Constituents tree** of the question, which contains phrase structure grammar representations of the words. For example, the question *Is Hirschsprung disease a mendelian or a multifactorial disorder?* produces the constituents tree shown in the Figure 4.4.

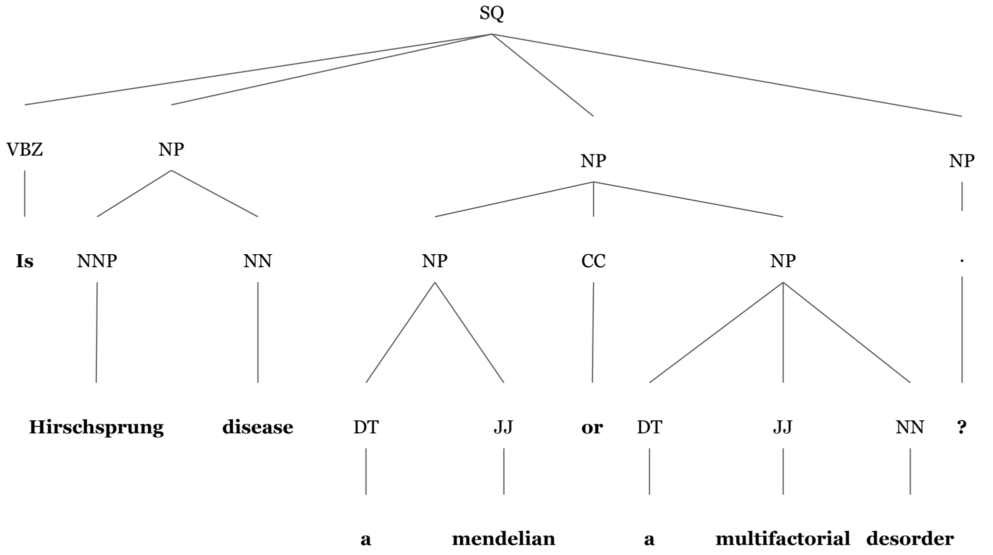


Figure 4.4: Example of constituents tree of the question *Is Hirschsprung disease a mendelian or a multifactorial disorder?*.

From the tree, we extracted the *constituents paths* concatenating the phrase constituents from the root to each leaf of the tree, which correspond to the words in the sentence. From the Figure 4.4 the following paths are produced:

'SQ/VBZ/Is',
'SQ/NP/NNP/Hirschsprung',
'SQ/NP/NN/disease',
'SQ/NP/NP/DT/a',
'SQ/NP/NP/JJ/mendelian',
'SQ/NP/CC/or',
'SQ/NP/NP/DT/a',
'SQ/NP/NP/JJ/multifactorial',
'SQ/NP/NP/NN/disorder',
'SQ/./?'

Finally, these paths are used as features to create the word embeddings vectors.

4.2.4 Dimensionality reduction of the embeddings

Because we had so many parameters to train in relation to the number of questions we had, we made experiments reducing the dimensionality of the pre-trained word vectors. We used ***Principal Component Analysis (PCA)***, to reduce the GloVe and Pyysalo embeddings. Originally of dimension 300 (GloVe) and 400 (Pyysalo), we reduced them to a 150-dimensional embeddings taking the first 150 principal components and preserving around 80% of the variance explained.

4.3 Architecture

This section is aimed to describe the different architectures we used to build the classifier. We are going to explain the two kinds of convolutional neural networks, that vary depending on the dimensions of the input matrices. The first version of the classifier is a 1-dimensional CNN with two types of inputs: n-hot encoding vectors and sentence-level embeddings. We also used a 2-dimensional CNN model over word-level embeddings, including our own-trained vectors and the pre-trained ones (GloVe, Pyysalo). Additionally, we used a linear model to measure the performance using n-hot encoding vectors.

4.4 1-dimensional CNN

The 1-dimensional neural networks are the ones that receive as input 1-dimensional vectors, like the n-hot encoded vectors and the sentence-level embeddings. We used Keras framework (Chollet et al., 2015) to build a 1-dimensional CNN composed by several convolutional layers interleaved by max-pooling layers, a drop out layer to reduce the number of trainable parameters and a final dense (fully connected) layer.

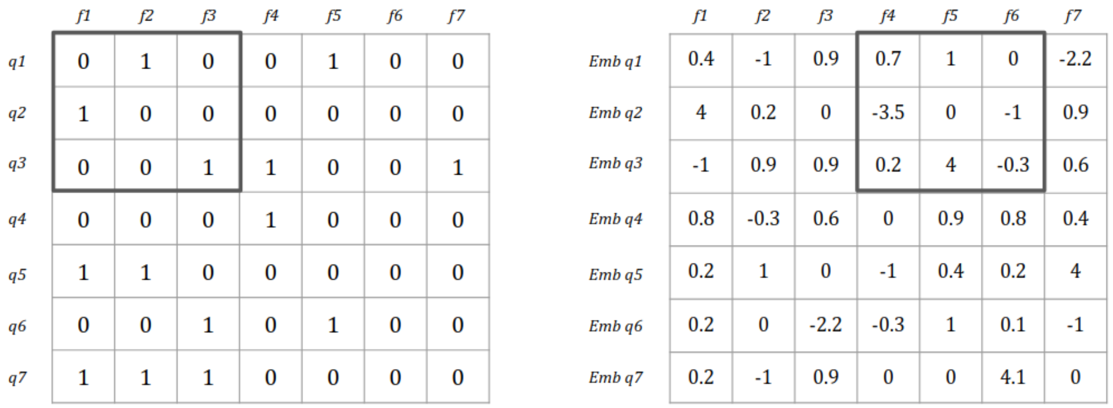
As we mentioned in the *Question Representation* section (4.2), when using the BOW approach, the input matrix consists on rows of n-hot encoding feature vectors representing the questions, forming a matrix with *number of questions* rows and *vocabulary size* columns (Figure 4.1).

When using embeddings, a non-trainable embedding layer is added at the top of the model. In this case, the input matrix is an array with *number of questions* embedding vectors of size of *embedding dimension.*

For the Convolutional layers, the filter matrix was initialized using the Glorot uniform initializer, that draws samples from a uniform distribution within [-limit, limit] where limit is $\sqrt{\frac{6}{(fan\_in+fan\_out)}}$ and where *fan_ in* is the number of input units in the weight tensor and *fan_ out* is the number of output units in the weight tensor (Chollet et al., 2015). The chosen activation function was the commonly used Rectified Linear Unit (ReLU).

In the 1D-CNNs a filter of dimension $n \times m$ applied to this kind of input matrices, takes n adjacent features from the same question in the horizontal axis and take features from m different sentences in the other dimension, see Figure 4.5. We fixed the number of steps (strides) in which the filter is shifted to 1 unit.

At the last step, we performed two types of classifications. The first, a **Multiclass** classification, where only the 4 general classes are predicted, and a **Multiclass-Multilabel** classification, where both, general and specific categories, are taken into account. In the Multiclass-Multilabel problem, every question is assigned to one of the four general categories and to one of the thirteen specific categories using the *sig-*



	f1	f2	f3	f4	f5	f6	f7
q1	0	1	0	0	1	0	0
q2	1	0	0	0	0	0	0
q3	0	0	1	1	0	0	1
q4	0	0	0	1	0	0	0
q5	1	1	0	0	0	0	0
q6	0	0	1	0	1	0	0
q7	1	1	1	0	0	0	0

(a) (3 × 3) filter for n-hot encoded matrix.

	f1	f2	f3	f4	f5	f6	f7
Emb q1	0.4	-1	0.9	0.7	1	0	-2.2
Emb q2	4	0.2	0	-3.5	0	-1	0.9
Emb q3	-1	0.9	0.9	0.2	4	-0.3	0.6
Emb q4	0.8	-0.3	0.6	0	0.9	0.8	0.4
Emb q5	0.2	1	0	-1	0.4	0.2	4
Emb q6	0.2	0	-2.2	-0.3	1	0.1	-1
Emb q7	0.2	-1	0.9	0	0	4.1	0

(b) (3 × 3) filter for sentence-level embeddings.

Figure 4.5: Example of (3 × 3) filters over n-hot encoded vectors matrix and sentence-level embedding vectors matrix.

moid activation function. In the Multiclass setting, a softmax is used as activation function over the neurons of the Dense layer. Figure 4.6 shows a single convolutional layer CNN architecture.

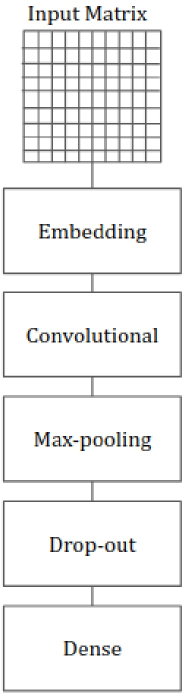


Figure 4.6: Example of a single layer CNN

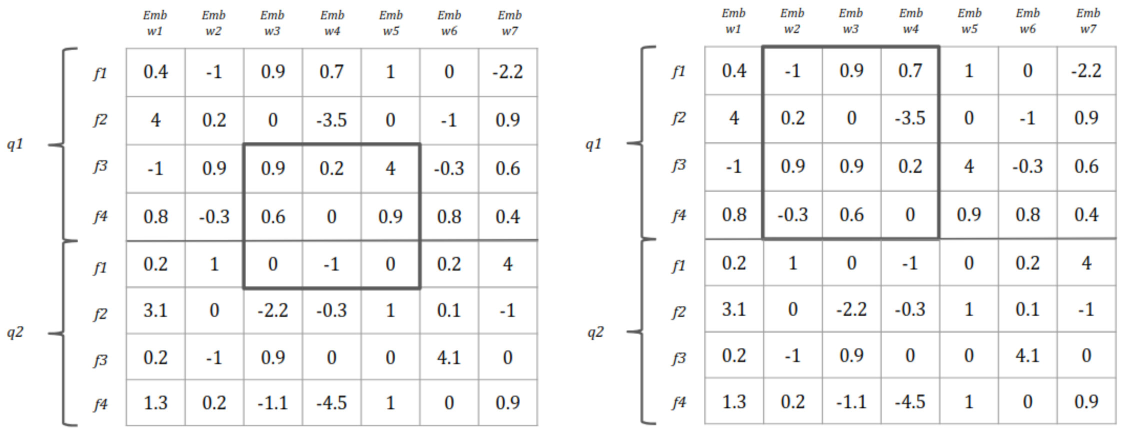
To train the model we used the *Categorical Cross entropy* loss function for the Multiclass scenario and *Binary Cross entropy* when doing Multiclass-Multilabel classification. Using binary cross entropy we treat each output label as an independent Bernoulli distribution. Additionally, we tried different optimizers, starting with

Stochastic Gradient Descent, and other adaptative learning rate algorithms like Adadelta, RMSprop and Adam.

4.5 2-dimensional CNN

The 2-dimensional CNNs receive 2-dimensional inputs, like word embeddings, which produce an input matrix of dimensions *number of questions* × *maximum sequence length* × *embedding dimension*, see Figure 4.2. With the exception of the input matrix, the architecture remains the same as in the 1-dimensional CNN.

In this case, a filter of size $(n \times m)$ corresponds to pick n adjacent word embedding vectors from the same question in the x-axis and m features in the y-axis. When moving the filter in the vertical dimension it could happen that the window contains features from more than one question, see Figure 4.7 (a). When $m = embedding\_dimension$, the filter occupies all the embedding vector, see Figure 4.7(b).



		Emb w1	Emb w2	Emb w3	Emb w4	Emb w5	Emb w6	Emb w7
q1	f1	0.4	-1	0.9	0.7	1	0	-2.2
	f2	4	0.2	0	-3.5	0	-1	0.9
	f3	-1	0.9	0.9	0.2	4	-0.3	0.6
	f4	0.8	-0.3	0.6	0	0.9	0.8	0.4
q2	f1	0.2	1	0	-1	0	0.2	4
	f2	3.1	0	-2.2	-0.3	1	0.1	-1
	f3	0.2	-1	0.9	0	0	4.1	0
	f4	1.3	0.2	-1.1	-4.5	1	0	0.9

(a) (3 × 3) filter for word-level embedding vectors matrix.

		Emb w1	Emb w2	Emb w3	Emb w4	Emb w5	Emb w6	Emb w7
q1	f1	0.4	-1	0.9	0.7	1	0	-2.2
	f2	4	0.2	0	-3.5	0	-1	0.9
	f3	-1	0.9	0.9	0.2	4	-0.3	0.6
	f4	0.8	-0.3	0.6	0	0.9	0.8	0.4
q2	f1	0.2	1	0	-1	0	0.2	4
	f2	3.1	0	-2.2	-0.3	1	0.1	-1
	f3	0.2	-1	0.9	0	0	4.1	0
	f4	1.3	0.2	-1.1	-4.5	1	0	0.9

(b) (3 × 4) filter (4 being the embedding vectors dimension) for word-level embedding vectors matrix.

Figure 4.7: Example of filters over word-level embedding vectors matrix.

4.5.1 Dependency matrix CNN

This model consists on creating a parallel set of features from the ***Dependency tree***, which contains the lexical dependencies between the words in the sentence, also known as grammatical relations. The idea is to retrieve the parent (in terms of dependency) for every word in the question. The following figure shows the dependency tree of the question *Is Hirschsprung disease a mendelian or a multifactorial disorder?*:

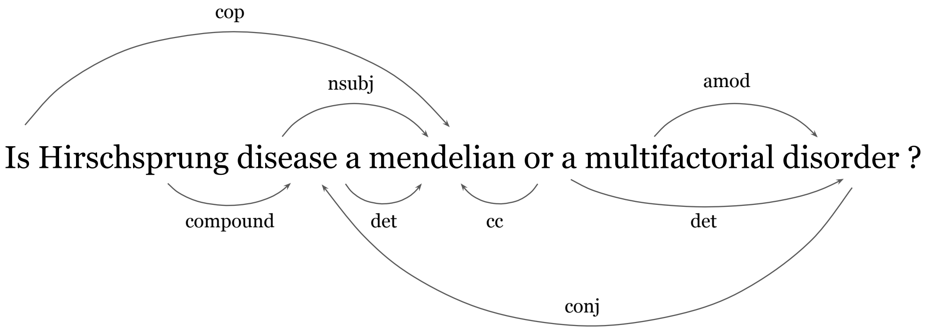
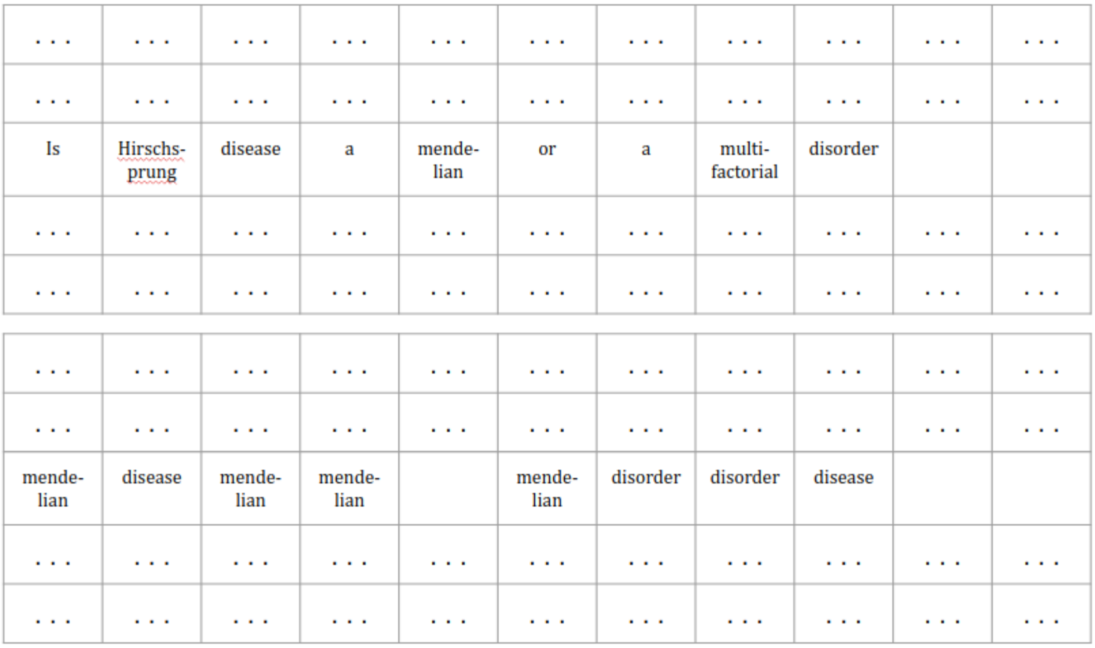

cop
nsubj
amod
Is Hirschsprung disease a mendelian or a multifactorial disorder ?
compound
det
cc
det
conj

Figure 4.8: Example of the dependency tree of the question *Is Hirschsprung disease a mendelian or a multifactorial disorder?*

Having the dependencies, we built a **dependency matrix** with the same dimension of the input matrix, but instead of containing n-hot encoded vectors or embeddings of the words in the sentence, contains n-hot encoded vectors or embeddings of the parent words. In Figure 4.9 we present a non-encoded input matrix and its corresponding dependency matrix. Both matrices are then transformed into word vectors representations and are passed simultaneously as input to the model.

In this case, the model is a 2D-CNN that receives two input matrices: the original one containing the words in the questions and the Dependency matrix that contains the parent words from the dependency tree. Additionally, each input matrix has a corresponding GloVe embedding matrix, that is also passed as input to build the respective embedding layers.

The output of each embedding layer is fed to an independent convolutional model that later will be combined using a Merge layer to take advantage of the dependency relations between the words. This merge is done using *sum*, *multiplication* or *average*. Finally, this Merge layer is used as input to the fully connected layer for classification (Multiclass and Multiclass-Multilabel). In the Figure 4.10 there is a representation of the dependency tree network.



...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...
Is	Hirschsprung	disease	a	mendelian	or	a	multifactorial	disorder		
...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...

...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...
mendelian	disease	mendelian	mendelian		mendelian	disorder	disorder	disease		
...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...

Figure 4.9: Example of the (non-encoded) input and dependency matrices of the question *Is Hirschsprung disease a mendelian or a multifactorial disorder?*

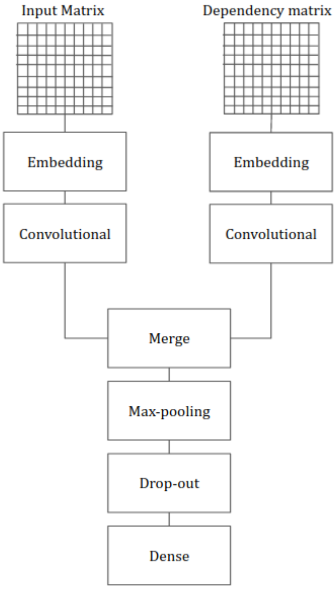


Figure 4.10: Example of the dependency tree CNN model

5 | Experiments and Results

In this chapter we are going to present the experiments we did and the results obtained using different models, different embeddings, multiple number of layers and varying sizes of filters. Additionally, we present the output using different optimizers and adding extra samples from Quora data set.

5.1 Experiments

In this section we are going to present the experiments we carried out and some of the configurations that we used. To begin with, we used a Multinomial Logistic Regression model with n-hot encoded vectors, that allows to perform mulitclass classifications. After that, we built 1D and 2D convolutional models and tuned the hyperparameters.

We built a 1D-CNN using n-hot encoding vectors and sentence-level embeddings. Then, we built a 2D-CNN using word-level embeddings as input. The number of the filters or the features to extract was set to 32 for both of the models after comparing the accuracy with higher and lower values. We used only one channel with squared filters of different sizes $(n \times n)$, with $n = \{2, 3, 5, 7\}$. In addition, we used non-squared filters that kept the aforementioned dimensions in the horizontal axis, but extended up to the embedding vectors dimension in the vertical axis. This translates to windows of size $(n \times 300)$ for GloVe, $(n \times 400)$ for Pyysalo and $(n \times 300)$ for our own-trained embeddings. For the Dependency Tree Model, which is also a 2D-CNN, we used three different kind of merge functions: sum, max and multiplication.

With respect to the number of layers, we started with a single convolutional layer and increased the number up to five layers looking for a change in the performance. We also added the interleaved max pooling layers and one drop-out layer.

As we mentioned in the 4.2 section, we built and used different kinds of embeddings: the GloVe and Pyysalo pre-trained embeddings and our own-trained ones. In the latter, we tried different representations. Initially we used the original version that includes the lemmas, POS tags and the description of the named entities, then, we used the ones which are obtained from the constituents tree paths.

Regarding to the optimization algorithm, we used Stochastic Gradient Descent and the adaptations RMSprop, Adam and Adadelta.

5.2 Results

In this section we present the results of the most relevant experiments. Even though the real order in which we carried out the experiments is not exactly the one presented below, we are going to discuss the results in a way that allow us to make the comparisons.

As a disclaimer, because the performance of a model changes based on the parallel choice of all the parameters, the results displayed in each table present the best performance obtained after the tuning. The comparisons are done changing one specific parameter and letting the rest fixed.

We used the *accuracy* as evaluation metric, which measures the overall performance of the classifier. However, because the accuracy does not allow us to measure the goodness of the model in predicting each single class, we used the Matthews Correlation Coefficient (MCC), that allows us to accomplish that. The MCC takes into account the true and false positives and true and false negatives, that makes it suitable for unbalanced classes, which is the case with our specific categories. We present the values of both metrics for the two problem settings: Multiclass and Multiclass-Multilabel.

We started comparing the different models with the different kind of embeddings and fixing all the rest of the parameters. In Table 5.1 we present the results for all of the configurations that we used. We can see that the Dependency model + GloVe and the 2D-CNN + GloVe had the best results in the Multiclass setting in terms of accuracy. Both also had the same moderately-good performance in terms of MCC. This means that adding information about the dependencies does not help to determine the type of the answer. The next models in the performance ranking, are the 2D-CNN + Word2Vec and 1D-CNN + Doc2Vec models, that had a low performance. Surprisingly, the Pyssalo model provided poor accuracy and MCC, which was comparable to a simple linear model with BOW.

In the Multiclass-Multilabel setting, we observed that the difference in accuracy among all the models was not big. Almost all of them, except for Pyysalo's, performed similarly good. Although, the Dependency model had a slightly better performance than the rest. On the contrary, all of the models presented bad results regarding the MCC. This lower performance was expected, because the combined number of classes sum up to thirteen, and apart from that, there was a high imbalance between the specific categories. On the other hand, since Pyysalo vectors had been trained with biomedical information, it was quite surprising that they provided such a bad performance.

Another interesting result is that all the algorithms performed better in the multiclass-multilabel case, even though it was a more challenging task. Since both groups of classes were manually created and the general classes were assigned by experienced PubMed curators, the quality of the annotations is not the differential factor. This means that the classifier is better at predicting specific categories, which addresses, not the type of the answer but the type of the content.

Model	Type of representation	Acc. (%) Multiclass/MCC	Acc. (%) Multiclass-Multilabel/MCC
Log. Regression	n-hot encoding	24 / 0.0	-
1D-CNN	n-hot encoding	30 / 0.0	86 / 0.04
1D-CNN	Doc2Vec embeddings	30 / 0.0	86 / 0.0
2D-CNN	Word2Vec embeddings	37 / 0.15	86 / 0.0
2D-CNN	GloVe embeddings	76 / 0.68	86 / 0.22
2D-CNN	Pyysalo embeddings	25 / 0.0	60 / 0.04
Dependency model	GloVe embeddings	76 / 0.68	91 / 0.27

Table 5.1: Comparison between models

Optimizer	Acc.(%) Multiclass/MCC	Acc.(%) Multiclass-Multilabel/MCC
SGD	71 / 0.68	86 / 0.09
Adadelta	76 / 0.68	87
Adam	72 / 0.63	89 / 0.28
RM-sprop	57 / 0.48	88 / 0.42

Table 5.2: Comparison among optimizers

Because of the reason that the Dependency Model had the best performance, we are going to present the rest of the results with this model.

After comparing the models, we measured the difference in performance using several optimizers. In the Table 5.2 we can see that the optimizers performed differently in each setting. The one that performed the best in the Multiclass setting was Adadelta and the one that performed the best in the Multiclass-Multilabel was Adam.

No study has been presented in the literature so far, that proves the superiority of a single optimizer over the others. Consequently, they are usually used as black boxes. Different optimizers perform differently depending on the scenario, so the fact that we obtained the best results with two different optimizers is not surprising.

In terms of the number of convolutional layers, we tried stacking from 1 to 5 layers to see whether there was an improvement or not. As the Table 5.3 shows, we did not find any increase in performance and therefore, we used the simplest (1 layer) architecture. This was also the case in other published works like *Kim et al. (2014)* and *Kalchbrenner et al. (2014)*.

Regarding the dimensions of the filter, we experimented with sizes of 2, 3, 5 and 7. We first used squared windows of $(n \times n)$ and then windows of $(n \times emb\_dim)$,

N. of Layers	Acc.(%) Multiclass/MCC	Acc.(%) Multiclass-Multilabel/MCC
1	76 / 0.68	86 / 0.22
2	73 / 0.67	86 / 0.16
3	74 / 0.65	86 / 0.0
4	75 / 0,67	86 / 0.0
5	60 / 0.44	86 / 0.0

Table 5.3: Comparison using different number of convolutional layers

Filter size	Acc.(%) Multiclass/MCC	Acc.(%) Multiclass-Multilabel/MCC
1×1	76 / 0.68	91 / 0.25
2×2	76 / 0.68	91 /0.27
5×5	76 / 0.68	91 / 0.27
7×5	76 / 0.68	90 / 0.29
2×300	77 / 0.69	87 / 0.25
3×300	73 / 0.67	88 / 0.29
5×300	74 / 0.69	88 / 0.29
7×300	75 / 0.66	88 / 0.27

Table 5.4: Comparison between dimensions of filters

where *emb_dim* is the dimension of the embeddings (300). We thought that taking more information into account (using $n \times emb\_dim$ filters), the accuracy would increase. However, from the Table 5.4 we can see that, contrary to what we have expected, using squared filters in the both settings provided a slightly better performance. Therefore, for the final model, we used squared (5×5) filters.

We did not notice any improvement in the performance of the network employing max-pooling and drop-out layers.

On a different matter, regarding to our attempt to incorporate new samples using POS tag patterns, we could not obtain good quality patterns that would be general enough to match unseen questions. We discovered that the maximum frequency in patterns that matched the conditions were of only two. Although we relaxed the conditions allowing to have patterns in more than one cluster, when we compared with unlabeled questions, the number of matches was still really low. This indicates that utilizing the POS tags solely cannot provide information about the structure of the question that we are dealing with.

On the contrary, we were successful incorporating new samples using the Quora data set. We compared the change in the performance with these extra samples and as we expected, adding a big amount of semi-supervised labeled questions increased the accuracy around 7%. This seems to indicate that our approach to label the Quora pair of questions worked well.

In terms of the dimensionality reduction using PCA, we did not see any improvement. However, on the contrary, we saw a slight downgrade in the performance of the classifier. Therefore we ended up using the original size of the vectors.

5.3 Analysis of the errors.

In this section we will describe a manual analysis of the errors in order to get insights for further improvement.

To perform the analysis we selected a set of 199 randomly extracted questions assigned to the four general classes.

The first step was to build the confusion matrix of our predictions, presented in Table 5.5. However, no interesting conclusions can be deduced from the confusion matrix. As expected, most of the cases (the positive ones) occur in the main diagonal and the the errors are distributed regularly. This means that the errors are probably due to multiple factors with no clear bias towards one of them. From the 199 questions of our set there are 58 errors, i.e. 29.14%.

True Class	factoid	summary	yesno	list	Total
Predicted Class					
factoid	33	9	5	2	49
summary	13	26	1	8	48
yesno	5	2	52	0	59
list	8	3	2	30	43
Total	59	40	60	40	199

Table 5.5: Confusion matrix

Next, we will discuss the main sources of errors presenting some examples.

- First, we noticed that in some cases the score of the true class is the second highest among our predictions and is close to the score of the predicted class. In 12 out of 58 cases, the difference between the two scores is less than 20%. Consider for instance the question *Is recommended the use of perioperative treatment with thyroid hormone therapy in patients undergoing coronary artery bypass grafting?*. The true class is *yesno* while the predicted class is *summary*. The predicted scores of the four classes are 0.0628961548, 0.4625069499, 0.454161793, 0.0204350203, corresponding to *factoid*, *summary*, *yesno*, and *list*, respectively. The distance between the true and the predicted class is just 0.0083451569. Even though the classifier failed, it did it by a very low margin. Maybe with more training data or more epochs these cases could be solved.
- Another issue detected is the length of the questions. As shown in Figure 5.3, there is a severe drop in accuracy (blue line) for short (less than 7 tokens) and

long (more than 16 tokens) questions. This drop is highly correlated with the number of examples to learn (gray line in the figure). This could be interpreted as an under-representation of these cases in the training set. Consider, for instance the 4-tokens question *Describe July Effect.*, a *summary* question incorrectly classified as *factoid.* This question contains only two significant words (no other case occurs in our sample set).

- *Stanford Core NLP library* is an excellent NL processor, but it has been built for processing general texts, no domain specific ones. In the case of extremely specific domains, as the medical/genetic one, the accuracy of the *Stanford Core NLP library* drops heavily. Several issues have been detected:
  - The case of multiword terms is specially challenging. Consider the case of *Fanconi anemia pathogenesis.* The tokenizer has produced [*Fanconi/JJ*, *anemia/NN*, *pathogenesis/NN*]. The correct split is [[*Fanconi anemia*], *pathogenesis*] but, besides the incorrect classification of *Fanconi* as an adjective (JJ), the three words were tokenized as different tokens and no multiword has been recognized. This issue results on a severe over-generation of cases that imply a drop on the learning accuracy.
  - The presence of acronyms and their long forms, sometimes with parenthesis, in other cases as appositions, also produces over-generation of cases. For instance, *What does polyadenylate-binding protein 4 (PABP4) bind to?* presents both issues.
  - Most of the named entities that occur in the questions (usually medical or genetic terms) are in the best case tagged as NNP (proper noun) because they are not found in the dictionaries, however, they are frequently erroneously tagged. The confusion of incorrectly assign the tags NN (singular noun) and NNS (plural noun), useful to distinguish a *factoid* from a *list*, is notably frequent and harmful. Why *sarcopenia/NNS* and *progeria/NN* are tagged differently in similar contexts is not clear at all.

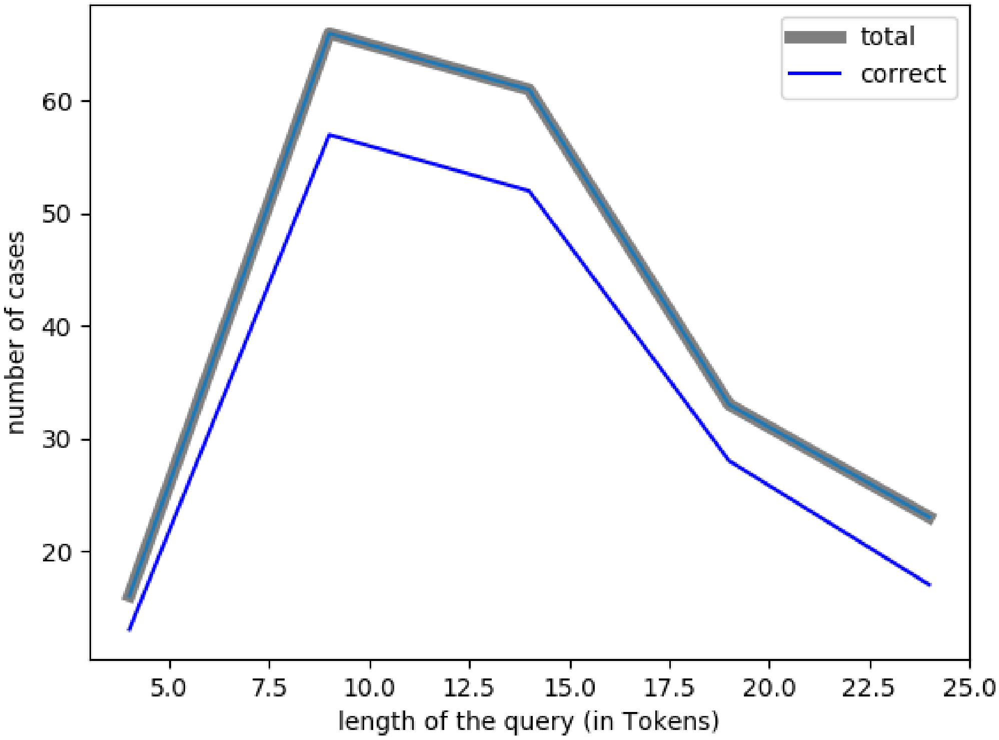


Figure 5.1: Histograms of length (in tokens) of questions

6 Conclusions, contributions and future work

In this project we built a model using the dependencies tree, obtaining an overall good classification performance using a CNN with a simple architecture. However, the Matthew correlation coefficient was specially low for Multiclass-Multilabel setting, which indicates that the performance of the classifier for less represented classes is inconsistent. In this regard, we noticed that the accuracy in the Multiclass-Multilabel problem was a lot higher compared to the classification in the four original classes. This could be due to the fact that the specific categories represent the theme of the questions and for that reason this problem involved topicalization of sentences. This seems to indicate that the model is better at detecting the topic of questions than predicting the expected type of an answer. In spite of that, we must mention that the design choice which improved the performance of the model more was the optimizer and the tuning of the learning rate.

Due to the fact that we found potential in incorporating information about the dependency relations between the words, for future works, we recommend the exploration of other configurations or extensions of the Dependency model. Apart from that, conducting experiments with different filter sizes simultaneously could also be promising.

On the other hand, due to the lack of domain specific question data sets, an automatic mechanism for detecting specific classes could be implemented using NLP techniques, e.g. Topic Models. This way, the manual tagging could be avoided. In the same spirit, using mechanisms to overcome the unbalance inside the classes would definitely improve the performance of the classifier.

We mentioned in section 5.1 that the method to increase the number of samples using the POS tags patterns did not provide good results. Perhaps, applying the patterns to other data sets could deliver better results. Apart from that, the use of wildcards in place of some POS tags could make it more general.

Bibliography

MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text, October 2013. URL `https://www.microsoft.com/en-us/research/publication/mctest-challenge-dataset-open-domain-machine-comprehension-text/`.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

M. Baroni, G. Dinu, and G. Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247. Association for Computational Linguistics, 2014. doi: 10.3115/v1/P14-1023. URL `http://www.aclweb.org/anthology/P14-1023`.

J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, 2013.

S. Bird, E. Klein, and E. Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

K. R. Biswas P, Sharan A. Question classification using syntactic and rule based approach. pages 1033–1038. 2014 International Conference on Advances in Computing. Communications and Informatics (ICACCI), 2014.

A. Bordes, S. Chopra, and J. Weston. Question answering with subgraph embeddings. *arXiv preprint arXiv:1406.3676*, 2014.

Y.-L. Boureau, J. Ponce, and Y. LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

E. Cabrio, C. F. Zucker, F. Gandon, A. Hallili, and A. Tettamanzi. Answering n-relation natural language questions in the commercial domain. In *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 1, pages 169–172, Dec 2015. doi: 10.1109/WI-IAT.2015.51.

M.-W. Chang, D. Goldwasser, D. Roth, and V. Srikumar. Discriminative learning over constrained latent representations. In *Human Language Technologies: The*

2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pages 429–437. Association for Computational Linguistics, 2010.

D. Chen, J. Bolton, and C. D. Manning. A thorough examination of the cnn/daily mail reading comprehension task. *CoRR*, abs/1606.02858, 2016. URL `http://arxiv.org/abs/1606.02858`.

D. Chen, A. Fisch, J. Weston, and A. Bordes. Reading wikipedia to answer open-domain questions. *CoRR*, abs/1704.00051, 2017. URL `http://arxiv.org/abs/1704.00051`.

F. Chollet et al. Keras. `https://keras.io`, 2015.

D. C. Ciresan, U. Meier, J. Masci, L. Maria Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237. Barcelona, Spain, 2011.

K. B. Cohen, D. Demner-Fushman, S. Ananiadou, Tsujii, and Junichi. Sigbiomed workshop on biomedical natural language processing. In *Proceedings of the 16th BioNLP Workshop*, 2017.

A. Conneau and D. Kiela. Senteval: An evaluation toolkit for universal sentence representations. *CoRR*, abs/1803.05449, 2018. URL `http://arxiv.org/abs/1803.05449`.

A. Conneau, G. Lample, M. Ranzato, L. Denoyer, and H. Jégou. Word translation without parallel data. *arXiv preprint arXiv:1710.04087*, 2017.

A. Deshpande. A beginnerś guide to understanding convolutional neural networks. `https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/`, 2016.

C. J. M. Dina Demner-Fushman, Wendy W. Chapman. What can natural language processing do for clinical decision. In *Journal of Biomedical Informatics 42 (2009) 760–772*, 2009.

D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, and J. Prager. Building watson: An overview of the deepqa project. In *AI magazine*, volume 31(3), pages 59–79, 2010.

T. G, B. G, M. P, P. I, Z. M, A. MR, W. D, K. A, P. S, P. D, A. Y, P. J, B. N, G. P, A. T, N. ACN, H. N, G. E, B.-A. L, S. M, A. I, and P. G. An overview of the bioasq large-scale biomedical semantic indexing and question answering competition. pages 1–28. BMC Bioinformatics. 2015, 2017.

I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

A. Hallili, E. Cabrio, and C. Faron-Zucker. Qalm: a benchmark for question answering over linked merchant websites data. 1272, 10 2014.

E. Hovy, U. Hermjakob, C.-Y. Lin, et al. The use of external knowledge in factoid qa. In *TREC*, volume 2001, pages 644–52, 2001.

K. A. Z. Imane Lahbari, Saïd El Alaoui Ouatik. A rule-based method for arabic question classification. pages 1–6. WINCOM 2017, 2017.

S. Kalaivani and K. Duraiswamy. Comparison of question answering systems based on ontology and semantic web in different environment. In *J. Comput. Sci. , vol. 8, no. 9, pp. 1407–1413, Sep. 2012.*, 2012.

N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

J. Ke, Y. Wang, and F. Xia. Question answering system with bi-directional attention flow. *CS224N Report*, 2017.

Y. Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014. URL `http://arxiv.org/abs/1408.5882`.

C. M. Kristina Toutanova, Dan Klein and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. pages 252–259. HLT-NAACL, 2003.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

X. Li, M. McGee-Lawrence, M. Decker, and J. Westendorf. The ewing,s sarcoma fusion protein, ews-fli, binds runx2 and blocks osteoblast differentiation. In *Journal of cellular biochemistry*, volume 111, 2010.

J. Lu, J. Yang, D. Batra, and D. Parikh. Hierarchical question-image co-attention for visual question answering. In *Advances In Neural Information Processing Systems*, pages 289–297, 2016.

R. P. Manning, Christopher D. and H. Schütze. *Introduction to Information Retrieval.* 2008.

G. L. Mikel Artetxe and E. Agirre. Learning principled bilingual mappings of word embeddings while preserving monolingual nvariance. *Proceedings of EMNLP, 2016*, 2016.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b.

M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 1003–1011, Stroudsburg, PA, USA, 2009. Association for Computational

Linguistics. ISBN 978-1-932432-46-6. URL `http://dl.acm.org/citation.cfm?id=1690219.1690287`.

S. Moen and T. S. S. Ananiadou. Distributional semantics resources for biomedical text processing. In *Proceedings of the 5th International Symposium on Languages in Biology and Medicine, Tokyo, Japan*, pages 39–43, 2013.

S. O. E. A. Mourad Sarrouti. A biomedical question answering system in bioasq 2017. pages 296–301. BioNLP 2017, 2017.

P. Nakov, D. Hoogeveen, L. Màrquez, A. Moschitti, H. Mubarak, T. Baldwin, and K. Verspoor. SemEval-2017 task 3: Community question answering. In *Proceedings of the 11th International Workshop on Semantic Evaluation*, SemEval '17, Vancouver, Canada, August 2017. Association for Computational Linguistics.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

X. Qiu and X. Huang. Convolutional neural tensor network architecture for community-based question answering. In *IJCAI*, pages 1305–1311, 2015.

D. Ravichandran and E. Hovy. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 41–47. Association for Computational Linguistics, 2002.

R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

S. Rosenthal, N. Farra, and P. Nakov. Semeval-2017 task 4: Sentiment analysis in twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 502–518, 2017.

S. Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL `http://arxiv.org/abs/1609.04747`.

S. H. Samuel L Smith, David HP Turban and N. Y. Hammerl. Offline bilingual word vectors, orthogonal transformations and the inverted softmax. *Proceedings of International Conference on Learning Representations, 2017*, 2017.

O. E. A. S. Sarrouti M. A machine learning-based method for question type classification in biomedical question answering. pages 209–216. Methods Inf Med. 2017, 2017.

R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

B. Y. Turian Joseph, Ratinov Lev. Word representations: A simple and general method for semi-supervised learning, 2010.

C. Unger, C. Forascu, V. López, A. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter. Question answering over linked data (QALD-5). In *Working Notes of CLEF 2015 - Conference and Labs of the Evaluation forum, Toulouse, France, September 8-11, 2015.*, 2015. URL http://ceur-ws.org/Vol-1391/173-CR.pdf.

D. H. D. Warren and F. C. N. Pereira. An efficient easily adaptable system for interpreting natural language queries. *Comput. Linguist.*, 8(3-4):110–122, July 1982. ISSN 0891-2017. URL http://dl.acm.org/citation.cfm?id=972942.972944.

C. Xiong, V. Zhong, and R. Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016. URL http://arxiv.org/abs/1611.01604.

W. Y. Xu J, Zhou Y. A classification of questions using svm and semantic similarity analysis. pages 31–34. Sixth International Conference on Internet Computing for Science and Engineering (ICICSE). 2012, 2012.

W.-t. Yih, M.-W. Chang, C. Meek, and A. Pastusiak. Question answering using enhanced lexical semantic models. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1744–1753, 2013.

Y. Yu, W. Zhang, K. Hasan, M. Yu, B. Xiang, and B. Zhou. End-to-end answer chunk extraction and ranking for reading comprehension. *arXiv preprint arXiv:1610.09996*, 2016.

Y. Zhang and B. C. Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *CoRR*, abs/1510.03820, 2015. URL http://arxiv.org/abs/1510.03820.

C. Zhou, C. Sun, Z. Liu, and F. C. M. Lau. A C-LSTM neural network for text classification. *CoRR*, abs/1511.08630, 2015a. URL http://arxiv.org/abs/1511.08630.

G. Zhou, T. He, J. Zhao, and P. Hu. Learning continuous word embedding with metadata for question retrieval in community question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 250–259, 2015b.