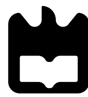
Universidade de Aveiro Departamento de Eletrónica, Telecomunicações e Informática
2019

Tiago Alexandre Melo Almeida

Recuperação de informação neural para resposta a perguntas biomédicas

Neural Information Retrieval for Biomedical Question-Answering



Universidade de Aveiro
2019

Departamento de Eletrónica,
Telecomunicações e Informática

Tiago Alexandre
Melo Almeida

Recuperação de informação neural para resposta a perguntas biomédicas

Neural Information Retrieval for Biomedical Question-Answering

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor Sérgio Guilherme Aleixo de Matos, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Doutor José Luis Guimarães Oliveira
professor associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Doutora Carla Alexandra Teixeira Lopes
professora auxiliar do Departamento de Engenharia Informática da Faculdade de Engenharia da Universidade do Porto

Doutor Sérgio Guilherme Aleixo de Matos
professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (orientador)

agradecimentos / **acknowledgements**

Quero aproveitar este momento para deixar um obrigado àqueles que me apoiaram, aturaram e ajudaram a crescer como estudante e pessoa ao longo destes cinco anos.
Aos meus pais agradeço a oportunidade que me deram.
Aos meus amigos mais próximos e colegas de curso agradeço especialmente o apoio, paciência e disponibilidade.
Um especial agradecimento ao meu orientador, Doutor Sérgio Matos, pela motivação, palavras encorajadoras e ajuda incansável. Finalmente agradeço os recursos computacionais cedidos pelo grupo bioinformática.
Por fim dedico esta tese à minha avó.

Palavras Chave redes neuronais, aprendizagem automática, recuperação de informação, recuperação de informação no dominio biomedico, modelos estatísticos.

Resumo Ao ritmo que a literatura biomédica publicamente disponível cresce, os sistemas de pesquisa atuais começam a ter dificuldades em manter um desempenho aceitável. Esta situação torna-se mais severa quando uma questão é submetida em linguagem natural. Movido por esta limitação, esta dissertação tem como principal objetivo criar um sistema automático de reposta a perguntas aplicado ao domínio biomédico que retorne, para uma dada questão, os documento mais relevantes e os seus respetivos excertos. O sistema foi dividido em três tarefas, a primeira consiste em encontrar documentos potencialmente relevantes para cada pergunta. No segundo passo, esses documentos são classificados por um modelo neural, que tem em consideração o significado e contexto da pergunta. Por fim, os excertos dos documentos relevantes mais significativos do ponto do vista do modelo neural são extraidos. Adicionalmente, foi proposto um novo modelo neural que é utilizado nas duas últimas tarefas do sistema. Como forma de validação, os resultados do sistema foram comparados com os resultados do desafio BioASQ deste ano, sendo que foi obtido o melhor resultado para o primeiro conjunto de teste e o terceiro melhor para o último conjunto de teste, enquanto que nos restantes os resultados ficaram próximos do topo.

Keywords

Neural networks, Deep learning, Information Retrieval, Neural Information Retrieval, Statistical Models.

Abstract

At the rate that publicly available biomedical literature grows, current searching systems start to struggle to maintain an acceptable performance. This situation becomes more severe when a question is submitted in natural language format. Moved by this limitation, this dissertation has the main purpose of creating an automatic question answering system applied to the biomedical domain that returns for a given natural language question, the most relevant documents and their relevant snippets. The system was divided into three steps, the first consist in finding potentially relevant documents to the query. In the second step, a more powerful deep neural model will rank these documents, in a way that the query context and meaning is taken into consideration. Additionally, it was been proposed a novel deep neural model that is used in the final two steps of the system. Finally, the snippets that helped the deep neural model to rank the most relevant documents are also extracted. As a way of validation, the system results were compared with the results from this year's BioASQ challenge, scoring the best result in first batch and third best on the last batch, while staying near to the top in the remaining batches.

Contents

Contents **i**

List of Figures **v**

List of Tables **ix**

Acronyms **xi**

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Objectives . . . 2

1.3 Contributions . . . 2

1.4 Dissertation outline . . . 3

2 Artificial Neural Networks and Deep learning **5**

2.1 Learning processes . . . 7

2.2 Supervised Tasks . . . 8

2.2.1 Classification Task . . . 8

2.2.2 Regression Task . . . 9

2.3 Regularization . . . 9

2.3.1 L2 regularization . . . 9

2.3.2 Dropout . . . 9

2.4 Convolutional Neural Network . . . 10

2.5 Recurrent Neural Network . . . 12

2.5.1 LSTM and GRU . . . 13

2.6 Attention mechanism . . . 15

2.6.1 Self-Attention . . . 17

3 Information Retrieval **19**

3.1 Ad-hoc Retrieval . . . 20

3.1.1 Vector Space Model and Cosine Similarity . . . 21

3.1.2 TF-IDF Weighting scheme . . . 22
3.1.3 BM25 Weighting scheme . . . 22
3.2 Neural Information retrieval . . . 23
3.2.1 Query-Document matching . . . 23
3.2.2 Query-Document ranking . . . 24
3.2.3 Word Representation . . . 25
3.2.4 Word2Vec . . . 26
3.2.5 Word2Vec subword extension . . . 29
3.3 Evaluation Metrics . . . 30

4 State of the Art **33**
4.1 Query-Document Matching . . . 33
4.1.1 Average word embedding . . . 33
4.1.2 Dual Embedding Space Model . . . 35
4.1.3 Deep Semantic Similarity Model . . . 36
4.1.4 Word Mover Distance . . . 38
4.2 Learning to Rank . . . 40
4.2.1 Deep Relevance Matching Model . . . 40
4.2.2 DeepRank . . . 45
4.2.3 A Hierarchical Attention Retrieval Model . . . 46
4.3 Summary . . . 49

5 Architecture and Implementation **51**
5.1 Technologies and libraries . . . 52
5.1.1 TensorFlow . . . 52
5.1.2 Numpy . . . 53
5.2 Fast retrieval . . . 53
5.2.1 Implementation of BM25 . . . 54
5.2.2 Implementation of AWE and AWE with TF-IDF . . . 58
5.2.3 Implementation of DSSM . . . 59
5.3 Neural ranking . . . 61
5.3.1 Implementation of DeepRank . . . 61
5.3.2 Implementation of Hierarchical Attention Retrieval Model . . . 68
5.4 Snippet extraction and visualization . . . 70
5.5 System as web application . . . 72

6 Experiments and Results **73**
6.1 Biomedical data . . . 73
6.1.1 BioASQ dataset . . . 73

6.1.2 PubMed . . . 74
6.2 Data preprocess . . . 75
6.2.1 Tokenization . . . 75
6.2.2 Collection and BioASQ statistics . . . 77
6.2.3 Embeddings . . . 78
6.3 Evaluation methodology . . . 78
6.3.1 Hardware . . . 78
6.4 Fast retrieval Results . . . 79
6.5 Neural Ranking Results . . . 82
6.5.1 DeepRank training behaviour . . . 83
6.5.2 Self-Attn-DeepRank training behaviour . . . 84
6.5.3 HAR training behaviour . . . 84
6.5.4 Neural models comparison . . . 85
6.6 Snippet Extraction Results - visualization . . . 87
6.6.1 DeepRank - inner working analysis . . . 89
6.7 BioASQ 7 taskB phaseA results . . . 90
6.8 System as web application . . . 91

7 Conclusion and Future work **93**

References **95**

A Appendix A: Visualization of more extracted snippets **101**

B Appendix B: Tensorflow graph visualization on TensorBoard **105**

C Appendix C: Implementation of the custom layers in *Keras* **111**

D Appendix D: Complete results of the BioASQ TaskB phase A **117**

List of Figures

1.1 The percentage of neural Information Retrieval (IR) papers at the ACM SIGIR conference - shows a growing trend of the neural approach for IR. The figure was taken from https://twitter.com/UnderdogGeek/status/1153280750889906176 2

2.1 Visual representation of an MLP architecture. Here is presented an MLP with three layers, input layer of size n, a hidden layer with m units and an output layer with k units. Each neuron has a bias value not shown in the figure. θ_1 and θ_2 matrices correspond to the weight value associated with connections between the neurons. 6

2.2 Visual representation, from the reference [8], of the dropout regularization, where the inactive units are represented by the red cross. In this Figure, the *bias weight* is represented by the connections of a neural unit that outputs 1, shown in yellow. 10

2.3 Workflow of a CNN. In a) Is shown the connection between the neural units. In b) is exemplified convolutional operation with 2×2 kernel, image was taken from [15]. At last in c) is shown the generic architecture of a CNN for a classification task, image was taken from [16] 11

2.4 Visualization of filters from different layers of the VGG-16 model [20], the filters activation's were removed from [21]. 11

2.5 Visual representation of the operations in CNN applied to a sentence. Image from [23]. . 12

2.6 Visual representation of a recurrent unit on the left and the same unit unrolled through time on the right. 13

2.7 Visual representation of an LSTM unit. 14

2.8 Simplistic example of the seq2seq model. Where the sentence "Bob is smart" is fed to the encoder and then the sentence "Bob é inteligente" is generated. 16

2.9 Simplistic example of the seq2seq with the attention mechanism. 17

2.10 Simplistic example of the self-attention mechanism. Here the self-attention is computed over an LSTM layer similar to the seq2seq-attention. 18

3.1 Generic representation of the major's components involved in a retrieval system. 20

3.2 Two-dimensional representation (only two terms are considered "gossip" and "jealous") of cosine measurement between the query vector and the document vector. The image was taken from [39]. 22
3.3 Two neural architectures used to implement the function $f(D, Q)$. In a) *Representation Based architecture*, b) *Interaction Based architecture*. The image was taken from [36]. . . 24
3.4 Visual illustration of the a) local representation and b) distributed representation. Circle represent each individual dimension and the painted circles represent a value, which in the case of local representation is 1. The image was taken from [42]. 26
3.5 Overview of the word2vec neural network architecture. W_{in} and W_out are the weight matrices, previously introduced as θ, i.e, these matrices correspond to the connections between the neural units. The image was withdrawn from the article [51]. 27
3.6 Overview of the Skip-Gram and Continuous Bag of Words approach. The image was withdrawn from the article [50]. 28

4.1 Two examples of passages retrieved for the query *Alburquerque*. In yellow, is represented the exact match and in green are the topical similar words. The image was withdrawn from the article [51]. 35
4.2 . The image was withdrawn from the article [60]. 37
4.3 . The image was withdrawn from the article [64]. 39
4.4 Architecture of the Deep Relevance Matching Model. The image was withdrawn from the article [67]. 41
4.5 Architecture of the document query interactions follow the ABEL-DRMM. The image was withdrawn from the article [68]. 44
4.6 Complete visualization of the deeprank model with several steps of granularity. The image was withdrawn from the article [69]. 46
4.7 Complete visualization of the HAR. The image was withdrawn from the article [72]. . . . 48

5.1 Overview of the principal modules of the proposed system. 51
5.2 Generic example of a *TensorFlow* computation on the left and the auto-differentiable operation on the write. From [75] 53
5.3 Flow diagram for the creation of the inverted index 55
5.4 On the left is presented the Flow diagram for the training process and on the right is presented the Flow diagram for the inference process 56
5.5 Tensor diagram of the dssm model in a). In b) tensor diagram created to train the dssm model. The blue color represents graph computations that were supported by the *Keras* API, the arrows are tensors. 60

5.6 DeepRank detection network. Q is the maximum number of query tokens, P is the maximum number of passages per query token, S is the maximum number of snippet tokens, and E is the size of the embedding vectors. The blue boxes correspond to native *Keras* layers. Correspondent *TensorFlow* graph can be visualized in Figure B.1. 62
5.7 DeepRank tensor diagram of the measure network. F number of filters and R output dimension of the GRU. The blue boxes correspond to native *Keras* layers. Correspondent *TensorFlow* graph can be visualized on Figure B.2 63
5.8 DeepRank tensor diagram of the aggregation network. The blue boxes correspond to native *Keras* layers. Correspondent *TensorFlow* graph can be visualized on Figure B.4 64
5.9 DeepRank tensor diagram. 65
5.10 DeepRank tensor diagram of the training architecture. 65
5.11 Snippet of tensor diagram of measure network with self-attention layer. 67
5.12 Tensor diagram of self attention layer. Correspondent *TensorFlow* graph can be visualized in Figure B.5 67
5.13 Tensor diagram of the complete HAR model. The blue boxes correspond to native *Keras* layers. Correspondent *TensorFlow* graph can be visualized in Figure B.6. 68
5.14 Tensor diagram of the Cross Attention Layer. Correspondent *TensorFlow* graph can be visualized in Figure B.7. 69
5.15 Tensor diagram of Hierarchical Attention Retrieval (HAR) of the training architecture . . 70
5.16 Overview of the web application components. 72

6.1 Comparison of the minimum number of occurrences of word for each tokenizer. 76
6.2 Distribution of the number of tokens by documents for both tokenizers. Histogram constructed with $bin = 500$ 77
6.3 Distribution of the number of tokens by queries for both tokenizers. Histogram constructed with $bin = 20$. 77
6.4 On the left, it is the evolution of the map score in the subset of the validation set and on the right, the evolution of the loss during training for the DeepRank model. 83
6.5 On the left, it is the evolution of the map score in the subset of the validation set and on the right, the evolution of the loss during training for the Self-Attn-DeepRank model. . . 84
6.6 Graphic. 85
6.7 Graphic with precision-recall curves at eleven recall levels for both DeepRank variations and BM25. The precision was computed over the 2500 documents that correspond to the complete ranking order produced by both neural models. 86
6.8 Graphic of MAP@N in function of different N for both DeepRank variations. 87
6.9 Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a true positive document at position 1 of the ranked list. 88

6.10 Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a false negative document that appears at position 3 of the ranked list. 88
6.11 Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a negative document that it is ranked at position 50. 89
6.12 Front page of the application on the left and search request on the right. Both images where captured with resolution of 640 × 360. 92
6.13 Display of the results for the previous search. The image was capture with resolution of 1366 × 768. 92

A.1 Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a positive document at position 2 of the resulting ranking list. Given the query *Are hepadnaviral minichromosomes free of nucleosomes?* 101
A.2 Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a negative document at position 1000 of the resulting ranking list. Given the query *Are hepadnaviral minichromosomes free of nucleosomes?* 102
A.3 Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a positive document at position 2 of the resulting ranking list. Given the query *What kind of analyses are performed with the software tool "unipept"* 102
A.4 Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a negative document at position 9 of the resulting ranking list. Given the query *What kind of analyses are performed with the software tool "unipept"* 102
A.5 Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a positive document at position 1 of the resulting ranking list. Given the query *"Are there any DNMT3 proteins present in plants?"* 103
A.6 Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a positive document at position 1000 of the resulting ranking list. Given the query *"Are there any DNMT3 proteins present in plants?"* 103

B.1 Tensorboard visualization associated with implementation of Figure 5.6. 105
B.2 Tensorboard visualization associated with implementation of Figure 5.7. 106
B.3 Tensorboard visualization of the CNN extraction sub-model 106
B.4 Tensorboard visualization associated with implementation of Figure 5.8. 107
B.5 Tensorboard visualization associated with implementation of Figure 5.12. 108
B.6 Tensorboard visualization associated with implementation of Figure 5.13. The output of a) and b) are fed to the self_attention and self_attention_2 layers presented in c). 109
B.7 Tensorboard visualization associated with implementation of Figure 5.14 110

List of Tables

2.1 Three alternatives to compute the attention weights. Here H is the dimension of the hidden state and A is a predefined dimension of the attention vector. 17

3.1 Five IR categories of tasks according to Onal *et al.* [36] 19

4.1 Most similar words to "eminem", using different embedding spaces. The data was withdrawn from the article [51]. 36

4.2 Summary table that compares all of the presented models in terms of task, embedding, overall architecture, if is required training, loss objective and the inference time. 49

4.3 Summary table with some comparatives results of some neural models 50

5.1 Hyperparameters chosen for the BM25 retrieval mechanism 57

5.2 Hyperparameters chosen for the DSSM 60

5.3 Hyperparameters chosen for the DeepRank 66

5.4 Hyperparameters chosen for the HAR model 70

6.1 Number of articles in 2018 MEDLINE/PubMed dump. 74

6.2 Size of the resulting vocabulary for the bllip and regex tokenizer 76

6.3 Size of the vocabulary with minimum token frequency restriction for the bllip and regex tokenizer 76

6.4 Server specifications 79

6.5 Record of empirical observations for different values of N 79

6.6 Fast retrieval models results for different values of N in the validation set. 80

6.7 Improvements of the AWE models with the tokenizer 3 over the validation set. 81

6.8 Comparison between the implemented BM25 ranking function and *elastic search* BM25 version. The average search time is measure in a total of 100 queries. 82

6.9 Additional information of the training hyperparameters for the DeepRank model 83

6.10 Additional information of the training hyperparameters for the Self-Attn-DeepRank model 84

6.11 Additional information of the training hyperparameters for the Hierarchical Attention Retrieval model 85

6.12 Evaluation of the neural retrieval models in the complete validation set 86
6.13 Slice of the expected interaction matrix for query terms *enzyme inhibited imetelstat* and snippet terms *imetelstat inhibited telomerase*. 90
6.14 Comparison with other submissions of the bioASQ task 7b phaseA results 91

D.1 Table with the results of the first test batch for the BioASQ TaskB phase A, the developed systems are highlighted in bold. 117
D.2 Table with the results of the second test batch for the BioASQ TaskB phase A, the developed systems are highlighted in bold. 118
D.3 Table with the results of the third test batch for the BioASQ TaskB phase A, the developed systems are highlighted in bold. 119
D.4 Table with the results for the fourth test batch of the BioASQ TaskB phase A, the developed systems are highlighted in bold. 119
D.5 Table with the results for the fifth test batch of the BioASQ TaskB phase A, the developed systems are highlighted in bold. 120

Acronyms

ADAM Adaptive Moment Estimation
ANN Artificial Neural Network
ABEL-DRMM Attention Based ELement-wise DRMM
AWE Average Word Embedding
BPTT Backpropagation Throughout Time
BoW Bag of Words
BM25 Best Match 25
CPU Central Processing Unit
CBoW Continuous Bag of Words
C-DSSM Convolutional Deep Structured Semantic Model
CNN Convolution Neural Network
CLSM Convolutional Latent Semantic Model
DAE Deep auto-encoder
DNN Deep Neural Network
DRMM Deep Relevance Matching Model
DSSM Deep Semantic Similarity Model
DSSM Deep Structured Semantic Model
D2Q Document-to-Query
DESM Dual Embedding Space Model
EMD Earth Mover Distance
GRU Gated Recurrent Units
GPU Graphics Processing Unit
HAR Hierarchical Attention Retrieval
ILSVRC ImageNet Large Scale Visual Recognition Competition
IDF Inverse Document Frequency
IR Information Retrieval
LSA Latent Semantic Analysis
LTR Learning to Rank
LSTM Long Short-Term Memory
LSTM-DSSM LSTM Deep Structured Semantic Model
MAP Mean Average Precision
MIP Mean Interpolated Precision
MRR Mean Reciprocal Rank
MSE Mean Squared Error
MLP Multi-Layer Perceptron

NLP Natural Language Processing
NEG Negative Sampling
NCE Noise Contrastive Estimation
nDCG Normalized Discounted Cumulative Gain
OOV Out-of-Vocabulary
POS Part-of-Speech
POSIT-DRMM POoled SImilariTy DRMM
Q2D Query-to-Document
QA Question Answering
RNN Recurrent Neural Network
RWMD Relaxed Word Moving Distance
Seq2Seq Sequence to Sequence
SG Skip-Gram
SIGIR Special Interest Group on Information Retrieval
SGD Stochastic Gradient Descent
TF Term Frequency
TF-IDF Term Frequency - Inverse Document Frequency
TREC Text REtrieval Conference
TPU Tensor Processing Unit
WMD Word Mover Distance

CHAPTER 1

Introduction

1.1 Motivation

Over the years, we witnessed the continued growth of biomedical literature. According to the US National Library of Medicine [1], almost 1 million biomedical articles are indexed by MEDLINE every year. This rate of growth poses a challenge to the biomedical experts, that need to routinely search a wide amount of scientific documents.

Indexing systems or most commonly known as search engines have emerged to help with this trend. These systems offer the users the ability to search the indexed documents, in an effective and efficient way. For example, PUBMED [2] is the most widely used search engine in the biomedical domain and has indexed over 24 millions of biomedical articles. However, as shown in the report of the BioASQ challenge [3][1], the experts still notice the following problems with this type of system:

- Search queries are not based in natural language, instead, it is usually used a combination of different keywords connected with boolean operators;
- Usually, these systems are unable to retrieve all the relevant documents;
- The experts must study all the retrieved documents until they find the desired information.

Currently, there has also been a growing interest in question answering systems since, what a user often wants is a precise answer to a question, instead of the full document [4]. Furthermore, Question Answering (QA) is a subfield of information retrieval, that specializes in producing or retrieving a single answer for a natural language question. Following this trend, biomedical question answering systems have also become more popular, mostly thanks

[1]More precisely, the results about question 6 "How useful are these search engines or tools? What are the main problems you face when using them?"

to the public competitions like BioASQ challenge [5], that push forward the research of QA systems applied to a biomedical domain.

In parallel, deep learning techniques are also growing. Especially after the paper by Hinton *et al.* [6] in 2006, on how to train a deep neural model, capable of achieving state-of-the-art precision in handwritten digit recognition. Since then, multiple neural models with excellent results were published. More recently this trend has reached the IR field, as shown in Figure 1.1 that represents the percentage of papers using neural models, that were accepted in the Special Interest Group on Information Retrieval (SIGIR)[2] conference.

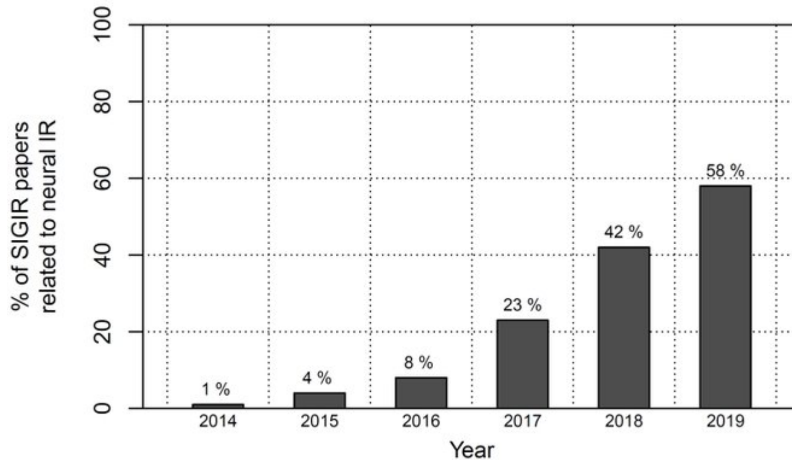


Figure 1.1: The percentage of neural IR papers at the ACM SIGIR conference - shows a growing trend of the neural approach for IR. The figure was taken from https://twitter.com/UnderdogGeek/status/1153280750889906176 .

1.2 Objectives

The main goal of this dissertation is to create an information retrieval system, applied to the biomedical domain, using deep learning techniques. As a way of evaluating and validating the system, the results were compared with the results from this year's BioASQ competition.

1.3 Contributions

This dissertation presents the following contributions to the neural information retrieval area:

- Implementation, in *TensorFlow*, of two state-of-the-art models that at time of writing are not available online;
- A novel deep neural model for information retrieval, that joins the *DeepRank* architecture with the self-attention mechanism, it is capable of achieving the same or better performance with half of the training weights of the *DeepRank* model;
- Best result in the first batch of the 2019 BioASQ challenge and third best on the last batch when compared with the submitted models.

[2]SIGIR website: http://sigir.org/

1.4 Dissertation outline

This dissertation is divided into seven chapters. The remaining chapters are organized as follows:

- Chapter 2 gives a quick overview of neural, convolution and recurrent networks and their inner workings. Other more advanced techniques are also introduced, such as the Attention mechanism.
- Chapter 3 introduces the information retrieval problem with some traditional methods and evaluation metrics. Then the core concepts and ideas of the new neural information retrieval trend are addressed.
- Chapter 4 shows the current neural information retrieval state-of-the-art models and for each model, an explanation and basic intuition is given about their inner workings.
- Chapter 5 presents the overall system, that is divided into a set of modules following a pipeline. Then, for each module, a set of requirements were defined, which guide its implementation.
- Chapter 6 shows the results for each module with a respective analysis. After that, the performance of the overall system is compared to the 2019 BioASQ challenge results.
- Chapter 7 summarizes the methodology during the system creation and adds some final conclusions about the results. In the end, some indications for future work and system evolution are referred.

CHAPTER 2

Artificial Neural Networks and Deep learning

This chapter intends to give a solid intuition about the working process of feed forward, convolutional and recurrent neural networks alongside with some more advanced techniques.

Humans always relied on nature as a source of inspiration. Birds inspired us to fly, the kingfisher shape improved the bullet train design, the model of wind turbines was inspired on humpback whales and much more examples can be found[1]. So, it seems only logical that the creation of an intelligent machine should be inspired by the structure of the human brain and this is the inspiration behind the Artificial Neural Network (ANN).

The ANN can be seen as an automatic pattern recognition framework belonging to the field of machine learning. An ANN is capable of automatically learning patterns (abstraction) from sparse and complex data, in order to solve a specific problem.

Nowadays, ANN can be applied to a vast list of problems from different areas such as Image Recognition, Machine Translation, Information Retrieval, Question Answering, Text-to-Speech and much more.

In a more detailed way, similar to the human brain, the ANN basic unit of computation is the neuron. These neurons can be organized in layers with different architectures. For example, Figure 2.1 shows a general architecture of a Multi-Layer Perceptron (MLP), that is the most basic type of an ANN. It has three layers, where each neuron of the previous layer is connected to all neurons of the next layer, also called a fully connected layer. Even though this network has only one hidden layer, it is possible to have more. Usually, if the ANN has a high number of hidden layers it is called Deep Neural Network (DNN).

[1]For more examples the following links can be used: https://www.digitaltrends.com/cool-tech/biomimicry-examples/ and https://www.bloomberg.com/news/photo-essays/2015-02-23/14-smart-inventions-inspired-by-nature-biomimicry

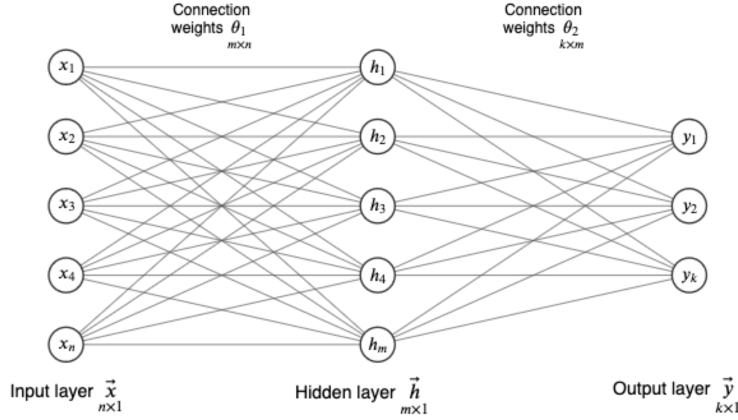


Figure 2.1: Visual representation of an MLP architecture. Here is presented an MLP with three layers, input layer of size n, a hidden layer with m units and an output layer with k units. Each neuron has a bias value not shown in the figure. θ_1 and θ_2 matrices correspond to the weight value associated with connections between the neurons.

Mathematically speaking, the computation performed by each unit (neuron) was introduced by McCulloch and Pitts [7] in 1943. From their definition and continuing with the example of Figure 2.1, the computation of the unit h_j can be presented by Equation 2.1.

$$h_j = \gamma\left(\sum_{i=1}^{m} \theta_{1_{(j,i)}} \times x_i + b_{1_j}\right) \tag{2.1}$$

This represents a linear combination between the weights ($\theta_{1_{(j,1,\dots,m)}}$) of each connection and the output of the previous neurons ($x_{1,\dots,n}$) plus a weight called *bias* ($b_{1_{(j)}}$) associated with each neuron. After this, a nonlinear function γ is applied, also known as *activation function*.

This formulation clearly reflects the inspiration from the human brain and the working process of the neurons: a human neuron receives electrical impulses (signals) from the connected neurons (linear combination) and when the number of incoming signals is greater than a threshold an output signal is fired (activation function).

However, using Equation 2.1 for computing the result of multiple neural units in a layer is inefficient. A better way would be to compute the result of the layer as a vector $\vec{h} = (h_1, h_2, ..., h_m)$, where each dimension corresponds to the output of a neural unit. This can be achieved with the help of linear algebra by vectorizing Equation 2.1, resulting in Equation 2.2.

$$\underset{m\times1}{\vec{h}} = \gamma\left(\underbrace{\underset{m\times n}{\theta_1} \cdot \underset{n\times1}{\vec{x}}}_{\text{dot product}} + \underset{m\times1}{\vec{b_1}}\right) \tag{2.2}$$

As shown previously, a neural unit or single layer of units is represented by a very simple computation and on its own can only solve linear problems. But when organized in a network

with multiple layers like in Figure 2.1, it is capable of approximating advanced mathematical functions, in order to solve extremely complex nonlinear problems. So, in the end, ANN can be mathematically formulated as a function F parameterized by a set of weight's θ, as shown in Equation 2.3.

$$F(x^n; \theta) \rightarrow \mathbb{R}^d \tag{2.3}$$

Here, x^n corresponds to a sample from the data, also designated input vector or feature vector of size n, which must be a numerical vector. The output is a real number vector of size d.

2.1 Learning processes

An ANN is capable of learning to solve a problem directly from the data with or without supervision. In the current literature, there are four types of supervision.

- **Supervised learning** - Supervision is given through a label, i.e, each sample from the dataset is associated with a correct prediction (label).
- **Unsupervised learning** - There is no supervision, i.e, the dataset is only composed of the individual samples.
- **Semi-supervised learning** - It is a combination of supervised and unsupervised learning, i.e, only a small set of samples are labeled.
- **Reinforcement learning** - Here the dataset is replaced by an interactive environment, that will be used to extract a supervised goal.

More formally, in supervised learning, all the samples x_i from a dataset are associated with a correct prediction y_i. So the objective is to find the optimal values for the weights θ, so that the ANN output, $\widetilde{y_i}$, is similar to the correct prediction y_i. From this description a mathematical optimization approach can be achieved by minimizing the error of the ANN predictions, i.e, the ANN training objective is to minimize the error (loss) between the y_i and $\widetilde{y_i}$ as shown in Equation 2.4.

$$J(\theta) = loss(y_i, \widetilde{y_i}) \Leftrightarrow J(\theta) = loss(y_i, F(x_i; \theta)) \tag{2.4}$$

Function $J(\theta)$ in Equation 2.4 represents the numeric value of the error with respect to function $F(x^m; \theta)$ (ANN predictions). The selection of the *loss* function depends on the type of problem that the network is trying to solve.

As mentioned before, the training of the ANN is reduced to an optimization problem and the most common method to solve this problem is based on algorithms of the gradient descent family. These follow an iterative approach, wherein each iteration the algorithm computes the gradients of $J(\theta)$, obtained from the partial derivatives of $J(\theta)$ with respect to the weights θ. Finally, the computed gradients are used to update the weights θ using a specific optimizer[2] such as Stochastic Gradient Descent (SGD), AdaGrad [9], AdaDelta [10] or Adaptive Moment

[2] Overview of these methods can be found in reference [8], pages 117-120 and 293-299

Estimation (ADAM) [11]. For neural networks, the computation of the gradients for each layer is achieved by the backpropagation algorithm [12].

$$\theta = Optimizer(\theta, \nabla J(\theta)) \tag{2.5}$$

Equation 2.5, describes the weight update process, depending on the chosen *optimizer*.

2.2 Supervised Tasks

Without loss of generality, when ANN is used in unsupervised or semi-supervised tasks, usually they are converted to a supervised task. This way the ANN can be optimized by the gradient descent algorithm. So in a supervised environment, the most common tasks that the ANN tries to solve are the classification and regression tasks.

2.2.1 Classification Task

Given a set of discrete classes $c \in C$ and $|C| > 1$, find a function capable of mapping the feature vectors $x_i \in Dataset$ to their correct class. The problem of image classification is an example of a classification task, where the features vectors x_i are the image pixels and the set of classes C are images classes (label). This function will learn the conditional probability $p(c|x_i)$ for all $c \in C$ given an input feature vector x_i.

In order to use an ANN to approximate this function, the output layer of ANN commonly has one neural unit for each class and the output of each unit is converted into probabilities using softmax operation. So the ANN is capable of outputting a probabilistic distribution of the input x_i over all the discrete classes c.

$$P(c = k|x_i) = \frac{e^{\vec{Z_{k,x_i}}}}{\sum_{j=1}^{|C|} e^{\vec{Z_{j,x_i}}}} \tag{2.6}$$

The softmax operation, Equation 2.6, creates a probabilistic distribution over the output layer of the ANN. The vector $\vec{Z_{k,x_i}}$ corresponds to network output (score) for the class k given x_i as input. Since the softmax uses a summation over all the classes scores, as shown in the denominator, this makes it a computationally expensive operation when the number of class ($|C|$) is high. In terms of optimization, usually, the objective is to minimize the *cross-entropy loss*, Equation 2.7.

$$loss(y_i, \widetilde{y}_i) = -\frac{1}{|Dataset|} \sum_{i=1}^{|Dataset|} \left(\underbrace{\vec{y_i}}_{C\times 1} \otimes \underbrace{log(\vec{\widetilde{y}_i})}_{C\times 1} \right), \\ where\ \widetilde{y}_i = P(c|x_i), \forall c \in C \tag{2.7}$$

Here, $\vec{y_i}$ correspond to the vector of the true probabilistic distribution over all the classes for the i-th sample in the Dataset. On the other hand, $\widetilde{y}_i$ represents the predicted probabilistic distribution by the neural network given the input feature of the i-th sample in the Dataset. The $\otimes$ symbol represents the element-wise multiplication between the two vectors. The average of the error for all the samples corresponds to the final value of the loss function.

2.2.2 Regression Task

Given a continuous real value y_i, find a function capable of mapping the feature vectors $x_i \in Dataset$ to a continuous variable $\widetilde{y_i}$, in such a way that y_i is close to $\widetilde{y_i}$ for the various samples.

In order to use an ANN to approximate this function, the output layer of ANN usually has only one neural unit without an activation function. So the output of this neural unit, $\widetilde{y_i}$, is directly used by the loss function. Equation 2.8 shows the commonly used Mean Squared Error (MSE) function.

$$loss(y_i, \widetilde{y_i}) = -\frac{1}{|Dataset|} \sum_{i=1}^{|Dataset|} \left(\vec{y_i} - \vec{\widetilde{y_i}}\right)^2 \tag{2.8}$$

2.3 Regularization

An ANN during the training can start to *overfit* the training data, which means that the network is approximating a function that is trying to fit every data point. So the network loses the capability of generalization to new data, i.e, when new data is fed to the network, it tends to perform poorly. The problem of *overfitting* usually occurs when the training data has few samples or when the neural network has a great number of trainable weights.

A solution to this issue is the applications of regularization techniques. This subsection will present *l2-regularization* and *Dropout* [13].

2.3.1 L2 regularization

The l2-regularization consist of applying l2-norm[3] as penalty term to the *loss function*, so rewriting Equation 2.4 in order to add the l2-norm, results in Equation 2.9.

$$J(\theta) = loss(y_i, F(x_i; \theta)) + \lambda \sum_{w \in \theta}^{|\theta|} w^2 \tag{2.9}$$

This regularization forces the network to chose small values for the weights (θ)[4] during the training process since big values will result in a higher loss. In other words, the neural network freedom is reduced by this regularization. Another important aspect is the hyper-parameter λ, which weighs the importance that is given to the regularization.

2.3.2 Dropout

Dropout was initially proposed by Hinton *et al.* [14] in 2012 and further detailed by Srivastava *et al.* [13]. For every training iteration, this technique will choose with a probability p, what are the neural units that are inactive, with the exception of the output layer. This idea is represented in Figure 2.2.

This simple idea presents a powerful solution to the *overfitting* problem. One possible interpretation of the potential of this technique is that in every training iteration a unique

[3] The l2-norm is equal to the squared Euclidean distance.

[4] Only the weight of the connection should be included in this regularization

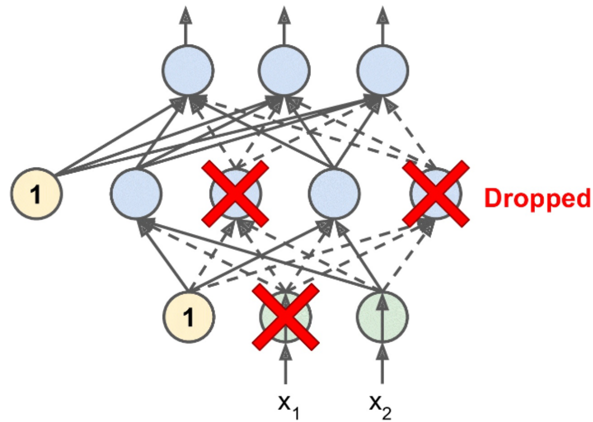


Figure 2.2: Visual representation, from the reference [8], of the dropout regularization, where the inactive units are represented by the red cross. In this Figure, the *bias weight* is represented by the connections of a neural unit that outputs 1, shown in yellow.

neural network is used. Considering that every neuron can be active or inactive, in fact, it is possible to have 2^N possible combinations of neural networks, where N is the total number of neural units that can be dropped.

2.4 Convolutional Neural Network

The Convolution Neural Network (CNN) is a special type of ANN, which is build using convolutional layers, hence its name. Unlike the fully connected layer shown initially, in a convolution layer the neurons of the current layer are only connected to a limited number of neurons in the previous layer, which is defined by the neural unit *receptive field*, as shown in Figure 2.3 image a). Each layer has a set of *filters*, also known as *kernels*, with the same size of neural units *receptive field* and a *bias* term per *filter*. The *filters* and *bias* are the trainable weights, so in each layer, all the neural units share the same weights, e.g, in image b) of Figure 2.3 is visible that all the neurons apply the same 2×2 *filter*. Thanks to this property the number of trainable weights is much smaller than a fully connected neural network with the same number of neural units. Finally, the output of a convolution layer is called *feature map* and the output of each neural unit is the same as presented before in Equation 2.1, with the difference that now the vector of weight is smaller (size of the *filter*) and each neuron performs this computation for every *filter*.

In a CNN architecture as shown in image c) of Figure 2.3, after the convolutional layer it is usually to see a pooling layer. This layer is simpler than a convolutional and has the purpose of reducing the size of the produced *feature maps*. In a pooling layer, each neural unit applies to its *receptive field* a simple aggregation operation, e.g, maximum, average or minimum. The most common aggregation operation is the maximum and when it is used the layer is called max-pooling layer. Another important aspect is that in this layer, the neural units do not have any *filters* or *bias*, so this layer does not have any trainable weights.

This type of neural network is extremely popular with tasks that involve image data, e.g., AlexNet [17], GoogleLeNet [18] and ResNet [19] are convolution neural networks that, in the

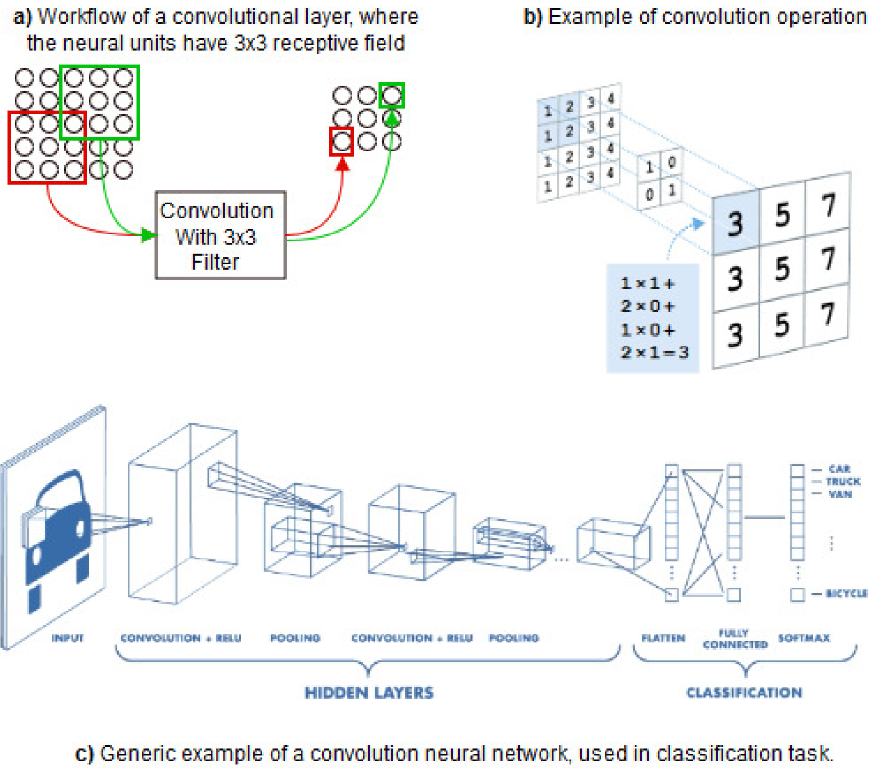


Figure 2.3: Workflow of a CNN. In a) Is shown the connection between the neural units. In b) is exemplified convolutional operation with 2×2 kernel, image was taken from [15]. At last in c) is shown the generic architecture of a CNN for a classification task, image was taken from [16]

years 2012, 2014 and 2015 respectively, achieved state-of-the-art results in the ImageNet Large Scale Visual Recognition Competition (ILSVRC) challenge.

The intuition behind the CNN is that the *filters* will learn to recognize small patterns over the input. For example, when using an image as input, the *filters* of the first convolution layer can learn how to recognize lines. From there, the *filters* of the second layer can learn how to recognize shapes and following this logic the *filters* of the following layers will learn more abstract representations. This idea is clearly visible in Figure 2.4.

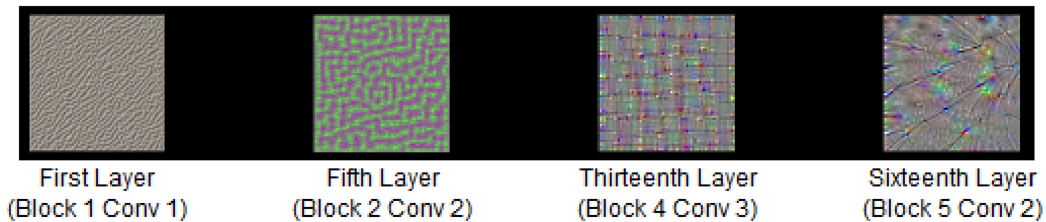


Figure 2.4: Visualization of filters from different layers of the VGG-16 model [20], the filters activation's were removed from [21].

After the success of the CNN in image related tasks, some works began to appear for text data, like in [22] and [23]. Generally, the application of a CNN to text data is straightforward,

as shown in Figure 2.5, the *filters* are applied directly to the words of a sentence. More precisely, the convolution operates over a numerical vector representation of the words, since the neural unit only works with numerical data, but this will be discussed more ahead. Usually, this type of convolution uses a kernel of n by e, where e is the dimension of the word numerical vector and is usually designated of *1D-convolution* since the kernel only moves along of the first dimension. Intuitively, the filters will identify n-grams and the pooling layer will extract the most important ones.

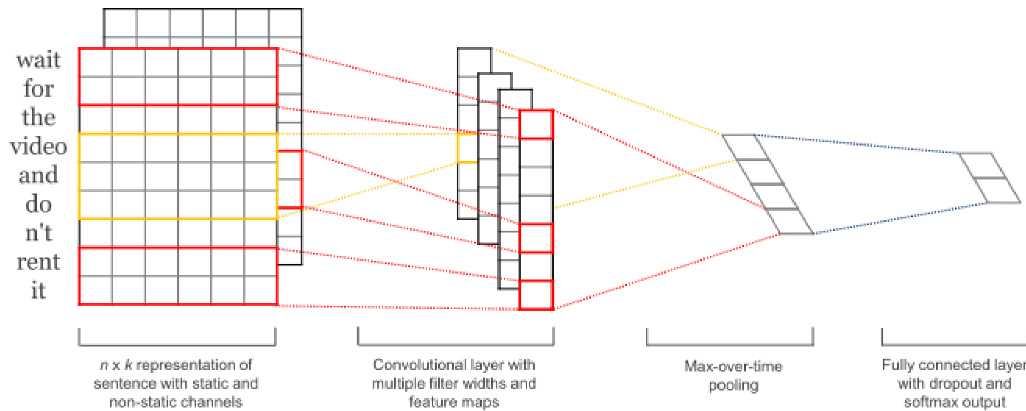


Figure 2.5: Visual representation of the operations in CNN applied to a sentence. Image from [23].

2.5 Recurrent Neural Network

The Recurrent Neural Network (RNN) is another type of neural network, that is specialized in handling sequential data of variable length. In this type of data, the samples ($\vec{x_{(t)}}$) are sequentially dependent of the previous samples ($\vec{x_{(t-1)}}, \vec{x_{(t-2)}}, ..., \vec{x_{(0)}}$), e.g, the next word of a sentence is dependent on the previous words or the next position of a moving car is dependent on its past positions. To handle this type of data an RNN uses recurrent units in its layers, instead of the normal neural unit.

In a more architectural view, the recurrent unit has one new connection that is pointing to itself, since it also uses the previously calculated output as input, as shown in the left side of Figure 2.6. This gives it the ability to remember and use the data from the past during the output calculation. In the right side of Figure 2.6, it is shown the unrolled view of the same neural unit through time.

Similar to Equation 2.2, a vectorized formulation of the recurrent layer computation is presented in Equation 2.10. The computation of the output layer for the time step t, $\vec{y_{(t)}}$, is given by a recursive function, with the terminal case when the time step t is equal to 0. The N is the number of neural units within the layer and M the number of input features. There are two sets of weights, θ_x and θ_y, the first one is associated with the input data $\vec{x_{(t)}}$ and the

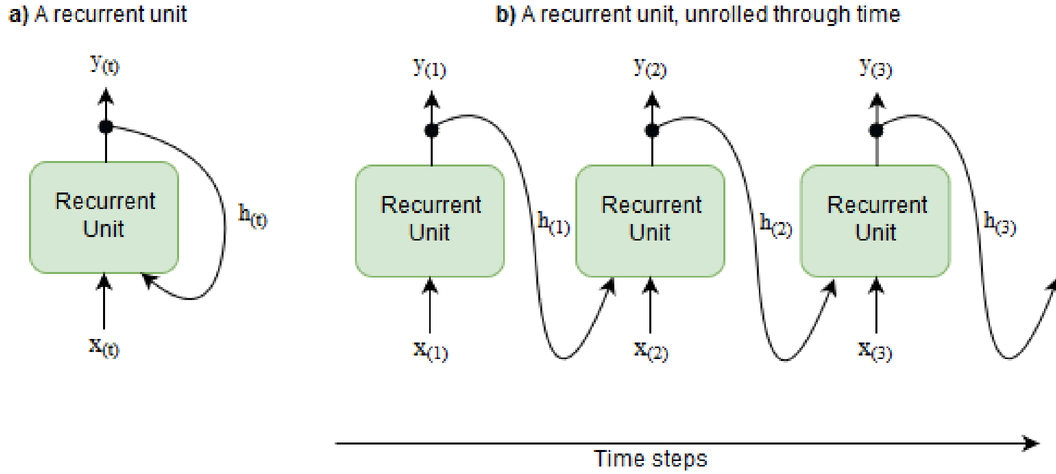


Figure 2.6: Visual representation of a recurrent unit on the left and the same unit unrolled through time on the right.

second one with the output of the previous time step $\vec{y_{(t-1)}}$.

$$\underset{N\times 1}{\vec{y_{(t)}}} = \begin{cases} \vec{0} & if\ t = 0, \\ \gamma\left(\underset{N\times M}{\theta_x} \cdot \underset{M\times 1}{\vec{x_{(t)}}} + \underset{N\times N}{\theta_y} \cdot \underset{N\times 1}{\vec{y_{(t-1)}}} + \underset{N\times 1}{\vec{b}}\right) & otherwise \end{cases} \tag{2.10}$$

The memory of this unit is essentially represented by the output of the previous step $\vec{y_{(t-1)}}$, sometimes also called hidden state $\vec{h_{(t-1)}}$, combined with the weights θ_y. The intuition here is that the weights during training will learn what the unit should remember from its last output, in order to minimize the neural network loss function.

For the training process, the gradients are calculated using the Backpropagation Throughout Time (BPTT) algorithm, which consists of applying the normal backpropagation algorithm to an unrolled throughout time RNN.

2.5.1 LSTM and GRU

A common problem with neural network training is the vanish or explosion of the gradients during the backpropagation algorithm, which is even more severe for the RNN. Unfortunately, the gradients tend to get smaller during the gradient propagation through the layers, which defines the vanishing problem. Or on the other hand, the gradients can also get bigger during the propagation, which characterizes the explosion problem. In 2010, Glorot and Bengio [24] published a paper to get a better understanding of these problems. Also in 2013, Pascanu *et al.* [25] presented the *Gradient Clipping* technique, that handles the explosion problem. This technique consists of clipping the gradients between a well defined interval and this way preventing them from exploding.

Besides the vanishing and explosion problem, the normal RNN unit also has difficult to remember long past events. So, more advanced recurrent units with the ability to retain long dependencies were proposed. The most popular unit is the Long Short-Term Memory (LSTM) and it was introduced in 1997 by Hochreiter and Schmidhuber [26]. Over the years, this unit

received several improvements, e.g., Zaremba *et al.* [27] show how to effectively apply dropout to the LSTM units to reduce *overfitting*.

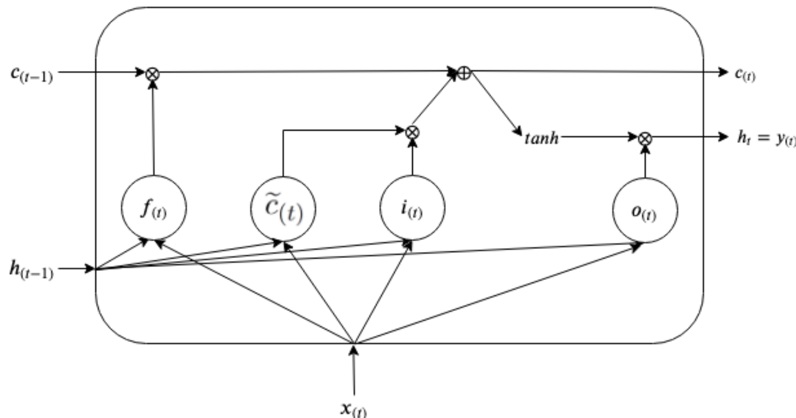


Figure 2.7: Visual representation of an LSTM unit.

The LSTM unit, as shown in Figure 2.7, uses a total of three gates, that enable it to chose what it should remember or forget from its internal memory, which also alleviates the gradient vanishes problem. In terms of memory, this unit has two vectors, the $\vec{c_{(t)}}$ long-term state and the $\vec{h_{(t)}}$ short-term state, that also corresponds to the layer output $\vec{y_{(t)}} = \vec{h_{(t)}}$.

The three gates that control the long-term state ($\vec{c_{(t)}}$) are implemented by a fully connected layer over the input and the previous state.

- **Input Gate** ($\vec{i_{(t)}}$) - Controls which part of the candidate memory ($\vec{\widetilde{c}_{(t)}}$) should be added to the long-term state ($\vec{c_{(t)}}$).
- **Forget Gate** ($\vec{f_{(t)}}$) - Controls which part of the long-term state ($\vec{c_{(t)}}$) should be erased.
- **Output Gate** ($\vec{o_{(t)}}$) - Controls which part of the long-term state ($\vec{c_{(t)}}$) should be output $\vec{y_{(t)}}$.

So, using vectorized notation, the computation of an LSTM layer $\vec{y_{(t)}}$ is given by Equation 2.11.

$$\underset{N\times 1}{\vec{y_{(t)}}} = \underset{N\times 1}{\vec{h_{(t)}}} = \underset{N\times 1}{\vec{o_{(t)}}} \otimes tanh\left(\underset{N\times 1}{\vec{c_{(t)}}}\right) \tag{2.11}$$

Where $\otimes$ represent the element-wise product between the output gate and the long-term state activated by the *hyperbolic tangent* function (tanh).

The long-term state ($\vec{c_{(t)}}$) is updated following Equation 2.12,

$$\underset{N\times 1}{\vec{c_{(t)}}} = \underset{N\times 1}{\vec{f_{(t)}}} \otimes \underset{N\times 1}{\vec{c_{(t-1)}}} + \underset{N\times 1}{\vec{i_{(t)}}} \otimes \underset{N\times 1}{\widetilde{c}_{(t)}} \tag{2.12}$$

where $\widetilde{c}_{(t)}$, corresponds to a candidate state memory, that is obtained from the inputs and the previous state as shown below,

$$\underset{N\times 1}{\widetilde{c}_{(t)}} = tanh\left(\underset{N\times M}{\theta_{xc}} \cdot \underset{M\times 1}{\vec{x_{(t)}}} + \underset{N\times N}{\theta_{hc}} \cdot \underset{N\times 1}{\vec{h_{(t-1)}}} + \underset{N\times 1}{\vec{bc}}\right) \tag{2.13}$$

Finally, the three gates are computed by

$$\begin{aligned}
\underset{N\times 1}{i_{(t)}} &= \sigma\left(\underset{N\times M}{\theta_{xi}} \cdot \underset{M\times 1}{\vec{x_{(t)}}} + \underset{N\times N}{\theta_{hi}} \cdot \underset{N\times 1}{\vec{h_{(t-1)}}} + \underset{N\times 1}{\vec{bi}}\right) \\
\underset{N\times 1}{f_{(t)}} &= \sigma\left(\underset{N\times M}{\theta_{xf}} \cdot \underset{M\times 1}{\vec{x_{(t)}}} + \underset{N\times N}{\theta_{hf}} \cdot \underset{N\times 1}{\vec{h_{(t-1)}}} + \underset{N\times 1}{\vec{bf}}\right) \\
\underset{N\times 1}{o_{(t)}} &= \sigma\left(\underset{N\times M}{\theta_{xo}} \cdot \underset{M\times 1}{\vec{x_{(t)}}} + \underset{N\times N}{\theta_{ho}} \cdot \underset{N\times 1}{\vec{h_{(t-1)}}} + \underset{N\times 1}{\vec{bo}}\right)
\end{aligned} \tag{2.14}$$

Every gate is described by computation similar to Equation 2.10, where the activation function is the sigmoid function (σ). So each gate will be represented by a vector, where each dimension will belong to the interval $]0, 1[$, where values close to zero will be ignored (closed gate) and values close to one will continue (opened gate). In the end, as described in Equations 2.11-2.14, the computation of $\vec{y_{(t)}}$ involves a series of sequential operations, including the update of the internal long-term state $\vec{c_{(t)}}$. This sequence of computations brings a clear downside in terms of performance when compared to a normal recurrent unit.

A light alternative to the LSTM is the Gated Recurrent Units (GRU). The GRU was proposed by Cho *et al.* in [28] and as shown by Greff *et al.* in [29] is a simplification of the LSTM, which it seems to perform just as well. More precisely the GRU is equivalent to set the *forget gate* of the LSTM to $\vec{f_{(t)}} = 1 - \vec{i_{(t)}}$.

2.6 Attention mechanism

Although RNN is capable of successfully learning the sequential dependencies in the data, its performance is always bound to the size of the memory representation. One simple example is the well known Sequence to Sequence (Seq2Seq) model [30]. This model aims to transform a sequential input into a sequential output and is widely adopted in translation tasks, e.g, given a sentence in English, the model will output a sentence in Portuguese with the same meaning. To accomplish this objective the model uses RNN-LSTM[5] as an encoder, to condense the meaning of the input sentence into a fixed size vector (hidden state of RNN). Then another RNN-LSTM, as a decoder, uses this inner representation in order to generate a new sentence (output sequence) with the same meaning. Figure 2.8 presents a simplistic visualization of the previously described model.

With respect to the previous model Bahdanau *et al.* [31] conjecture that the encoded vector is a bottleneck of these types of models since the encoder must be able to compress all the important information in the input sentence into a fixed size vector. This bottleneck is more severe for long sentences, especially when the sentences during training are shorter than sentences during test/production.

To cope with this issue Bahdanau *et al.* [31] proposed an attention layer, where the decoder will be able to see a weighted representation of all the input. That is, in each timestep, t, the decoder will get as input the decoder previous state, $\vec{h_{(t-1)_d}}$, and a context vector, $\vec{c_{(t)}}$. In Figure 2.9 is presented the introduction of the attention mechanism to the Seq2Seq model.

[5]This notation means that the recurrent neural network is build using lstm layers

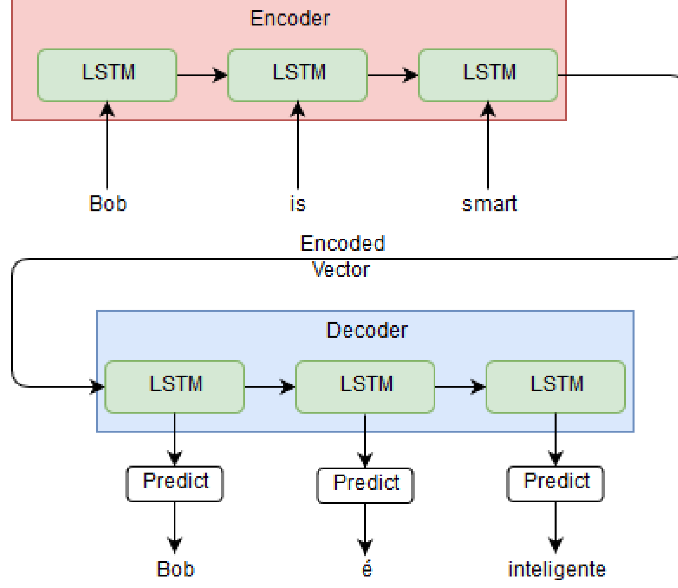


Figure 2.8: Simplistic example of the seq2seq model. Where the sentence "Bob is smart" is fed to the encoder and then the sentence "Bob é inteligente" is generated.

Formally speaking, given the encoder hidden states, $\vec{h_{(t)}}_e$, for each timestep, t, the context vector, $c_{\vec{(t)}}$, is computed as the weighted sum over these hidden states as shown in Equation 2.15,

$$\underset{N\times 1}{\vec{c}_{(t)}} = \sum_{j=1}^{t} a_{ij} \times \underset{N\times 1}{\vec{h_{(j)}}_e} \tag{2.15}$$

where the attention weights a_{ij} are obtained by the *alignment model* (*align*) and normalized with softmax operation, Equation 2.16.

$$\begin{aligned} a_{ij} &= softmax\left(align\left(\vec{h_{(i-1)}}_d, \vec{h_{(j)}}_e\right)\right) \\ a_{ij} &= \frac{e^{align\left(\vec{h_{(i-1)}}_d, \vec{h_{(j)}}_e\right)}}{\sum_{k=1}^{t} e^{align\left(\vec{h_{(i-1)}}_d, \vec{h_{(k)}}_e\right)}} \end{aligned} \tag{2.16}$$

Here the *alignment model* uses the encoded inputs, $\vec{h_{(j)}}_e$, and the previous state of the decoder, to calculate a matching score that indicates how well the inputs around position j will be related with the output at position i. So intuitively the context vector c_t inherits the responsibility of storing the important information in each timestep.

In terms of the *alignment model*, Bahdanau *et al.* used a feed forward neural network, but during the years other *alignment models* were proposed. Table 2.1 shows other implementations for the *alignment model* (*align*).

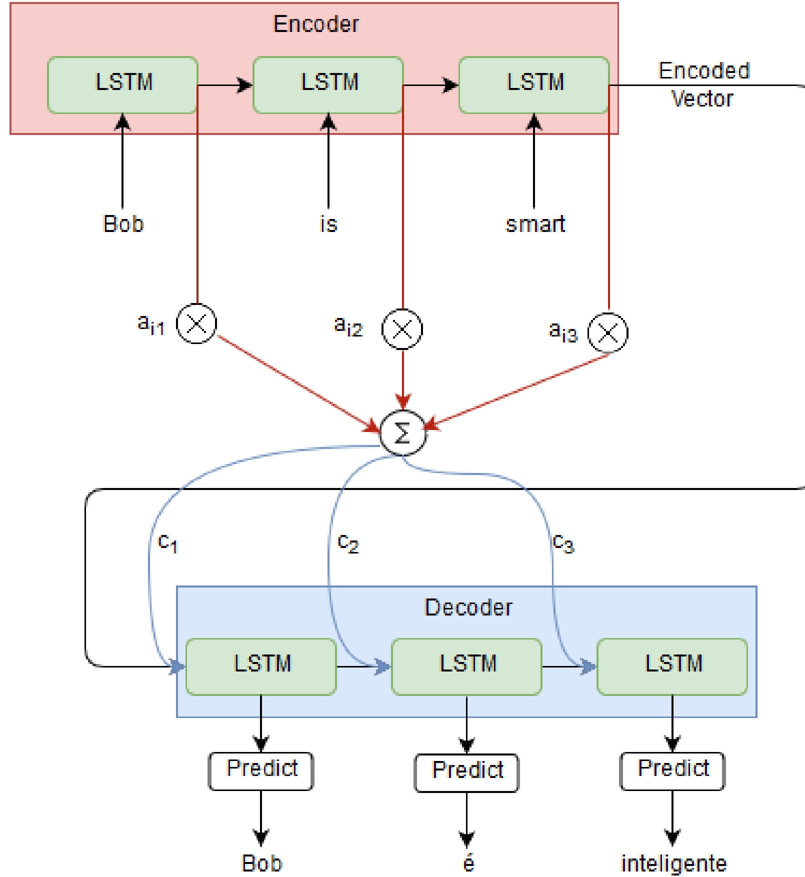


Figure 2.9: Simplistic example of the seq2seq with the attention mechanism.

Table 2.1: Three alternatives to compute the attention weights. Here H is the dimension of the hidden state and A is a predefined dimension of the attention vector.

Alignment model	Formulation	Authors
Feed Forward Net	$\underset{1\times A}{v^T} \cdot tanh\left(\underset{A\times 2H}{W} \cdot \underbrace{\{\underset{H\times 1}{\vec{h}_{(i-1)_d}}; \underset{H\times 1}{\vec{h}_{(j)_e}}\}}_{\text{concatenation}}\right)$	Bahdanau *et al.* [31]
Dot-product	$\underset{1\times H}{\vec{h}_{(i-1)_d}}^T \cdot \underset{H\times 1}{\vec{h}_{(j)_e}}$	Luong *et al.* [32][6]
Scaled dot-product	$\frac{\underset{1\times H}{\vec{h}_{(i-1)_d}}^T \cdot \underset{H\times 1}{\vec{h}_{(j)_e}}}{\sqrt{n}}$	Vaswani *et al.* [33]

2.6.1 Self-Attention

Self-Attention presents a way of applying the attention mechanism only to the input sequence, i.e, the *attention model* only uses the input sequence. The intuition is to create a better representation of the input sequence by giving more weight to individual inputs that are more important to the respective task.

Lin *et al.* [34] shown a recent implementation of the self-attention mechanism with the objective of generating a vector representation for a sentence. Based on the visualization

presented in Figure 2.10, this self-attention mechanism is similar to the previously attention mechanism, where the context vector c_t becomes a vector representation of the sentence and the attention weights, a_{ij}, are computed based on the Equation 2.16. The different *alignment models* presented in Table 2.1 are also theoretically applicable to this type of attention where $\vec{h_{(i-1)_d}}$ is discarded or replaced by a trainable vector.

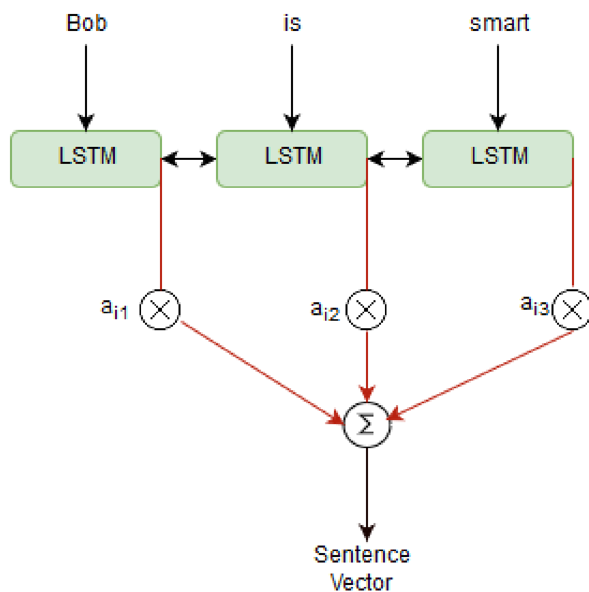


Figure 2.10: Simplistic example of the self-attention mechanism. Here the self-attention is computed over an LSTM layer similar to the seq2seq-attention.

CHAPTER 3

Information Retrieval

This chapter aims to give some background about the information retrieval field and introduce the core idea in neural information retrieval.

As mentioned in the literature [35], the meaning of the term *information retrieval* can be very broad. However as an academic field of study *information retrieval* can be defined as:

> *Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).*

Given the wide universe of different tasks in IR, Onal *et al.* [36] chose to divide them into five categories, as shown in Table 3.1. In bold are the tasks that will be the focus of this dissertation, **Ad-hoc retrieval** and **Question Answering**.

Table 3.1: Five IR categories of tasks according to Onal *et al.* [36]

Tasks	More granular example
Ad-hoc retrieval	**Text retrieval** **Document ranking** Query expansion Query re-weighting Product search
Question Answering	**Answer sentence retrieval** Conversational agents
Query understanding	Query suggestion Query auto completion Query classification
Similar item retrieval	Related document search Detecting text re-use Content-based recommendation
Sponsored search	

.

3.1 Ad-hoc Retrieval

Continuing with the literature [37], the following quotation defines the *ad-hoc retrieval* task and other important terminology.

> ***Ad-hoc retrieval*** *is the most standard IR task. In it, a system aims to provide documents from within the collection that are relevant to an arbitrary user information need.*
>
> ***Information need*** *is the topic about which the user desires to know more.*
>
> ***Query*** *is what the user conveys to the computer in an attempt to communicate the information need.*
>
> ***Relevant document*** *A document is relevant if it is one that the user perceives as containing information of value with respect to their personal information need.*

Furthermore, for an *ad-hoc retrieval* task a system should be able to present, usually in a ranking order, the most relevant documents that were retrieved from a large collection, given a user query. This query expresses the information that the user is searching for.

Since the implementation of each system is dependent on the retrieval task, Figure 3.1 shows the architectural majors steps that a retrieval system usually performs. For this dissertation only the study of the *Retrieval mechanism*, green box in Figure 3.1, will be considered, giving less emphasis to the other steps. However, more detailed information about the other steps can be found in reference [38] Chapters 2 (text preprocessing), 4 and 5 (index).

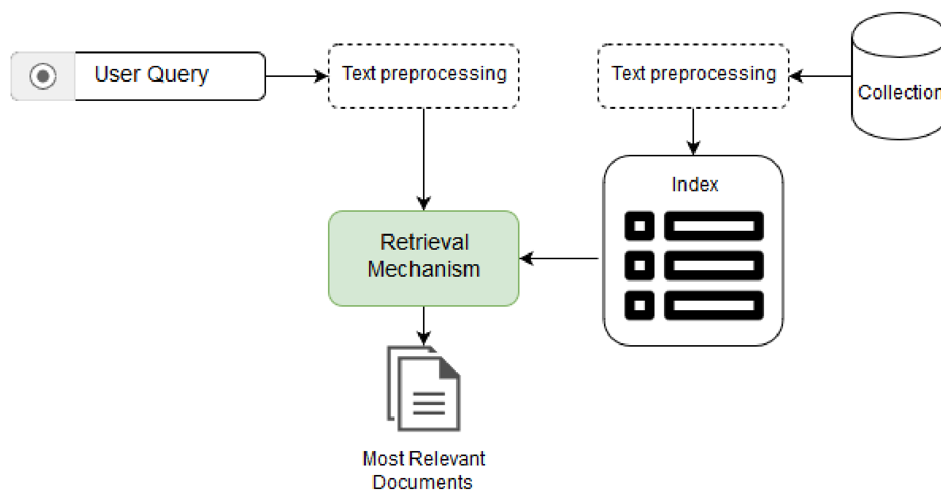


Figure 3.1: Generic representation of the major's components involved in a retrieval system.

The most simpler *Retrieval mechanism* is the boolean model, that is based on set theory and boolean logic. In this model, each word is considered as a term, each document as a set of terms and the query is represented as a boolean expression of the terms. The documents that satisfy the query expression are retrieved to the user.

A major drawback of this model is that it does not use any knowledge about the words and considers that every word has the same importance, thus neglecting rare words.

3.1.1 Vector Space Model and Cosine Similarity

An alternative to the boolean model is to assign a weight to each term, in a way that the most important/relevant terms have a bigger weight. In this perspective, each query or document is represented as a vector, where each dimension corresponds to a term and its respective weight. This representation of documents and queries as vectors in the same dimensional space is known as *Vector Space Model*.

In this model, the retrieval is achieved by computing the similarity between the query vector and all the document vectors and the documents with the higher scores are retrieved. The cosine similarity, Equation 3.1, is the most used measurement of similarity between vectors. This measures the cosine of the angle between two non-zero vectors, a value in the interval $[-1, 1]$.

$$cos(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{\|\vec{q}\| \left\|\vec{d}\right\|} \tag{3.1}$$

Here, the numerator is the inner product between the two vectors, also called dot product. The denominator corresponds to the length-normalization of the vectors into a unit vector, that is achieved by their Euclidean norms ($\|.\|$). Since the query vector must be compared with all the document vectors, Equation 3.1 can be optimized to directly compute all the scores using linear algebra, resulting in Equation 3.2.

$$cos(\vec{q}, C) = \frac{\overbrace{\underset{D\times V}{C} \cdot \underset{V\times 1}{\vec{q}}}^{D\times 1}}{\underbrace{\underbrace{\|C\|}_{D\times 1} \underbrace{\|\vec{q}\|}_{1\times 1}}_{D\times 1}} \tag{3.2}$$

Here, matrix C correspond to all the documents in the collection, where each row correspond to a document vector, so D is the total number of documents and V is the dimension of the vector, i.e, the total number of unique terms in the collection. The Euclidean norm of C is computed along the row dimension. Finally, C and $\|C\|$ can be already computed and stored in cache.

Using Figure 3.2 as an illustrative example, the query q is represented in a two-dimensional space as a vector $\vec{v}(q)$. Given three documents d_1, d_2, d_3 the most relevant to that query, according to the cosine similarity, is the document two (d_2), since the angle between q and d_2 is the closest to zero, which mean that the cosine of this angle is the closest to one, when comparing with d_1 and d_3.

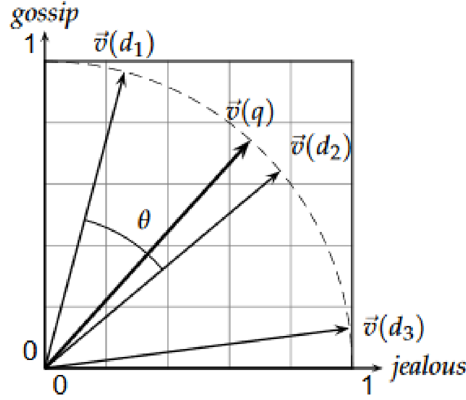


Figure 3.2: Two-dimensional representation (only two terms are considered "gossip" and "jealous") of cosine measurement between the query vector and the document vector. The image was taken from [39].

3.1.2 TF-IDF Weighting scheme

Term Frequency - Inverse Document Frequency (TF-IDF) is an example of a weighting mechanism, that weights each unique term present on a query or document given its importance. So each query or document is mapped to a vector with the size of the vocabulary and the value of each dimension is given by the TF-IDF weight. This formulation is presented in Equation 3.3.

$$tfidf_{t,d} = tf_{t,d} \times idf_t \tag{3.3}$$

Term Frequency (TF) ($tf_{t,d}$) captures the importance of each term with respect to that document and can be directly represented by the number of occurrences of term t in the document (d). On the other hand, Inverse Document Frequency (IDF) idf_t captures the importance of rare terms according to the collection, calculated as shown in Equation 3.4.

$$idf_t = log\left(\frac{D}{df_t}\right) \tag{3.4}$$

Here, D correspond to the total number of documents and df_t stands for *document frequency*, usually represented by the number of documents that contain the term (t).

3.1.3 BM25 Weighting scheme

The Best Match 25 (BM25) belongs to the family of weighting schemes implemented by the Okapi retrieval system, presented at the Text REtrieval Conference (TREC) by Robertson *et al.* [40]. In their article, the authors showed the different weighting schemes BM1, BM11/BM15 and BM25, that were used, respectively, in the first, second and third edition of the TREC competition.

As shown in [41], BM25 to take into consideration the term frequency and the length of the document in a more sensitive way.

$$score(q,d) = \sum_{t_q \in Q} \overbrace{\underbrace{idf_t}_{\text{first part}} \times \underbrace{\frac{(k_1+1)tf_{t,d}}{k_1(1-b+b(|D|/avg_{|D|}))+tf_{t,d}}}_{\text{second part}} \times \underbrace{\frac{(k_3+1)tf_{t,q}}{k_3+tf_{t,q}}}_{\text{third part}}}^{weight(q,d)} \tag{3.5}$$

In Equation 3.5 is shown a ranking function that uses the BM25 weighting scheme. In the first part, it takes into consideration the IDF of a term. In the second part weights the term frequency in a document, $tf_{t,d}$, with the document length $|D|$ and the average document length of the collection $avg|D|$. At last, the third part weights the length of the query by taking into consideration the frequency of a term in a query, this part tries to alleviate the penalization for long queries. The k_1, k_3 and b are hyperparameters that can be fine tuned to the data collection.

3.2 Neural Information retrieval

In general, traditional approaches to IR tasks rely on high engineering techniques to perform the retrieval task. An example is the BM25 weight mechanism, that uses a well defined equation to calculate the term-document weight based on word statistics, which works well for tasks that require an exact match between query and document terms. However, it will fail to retrieve semantically similar documents, since this and other types of more traditional techniques do not capture any knowledge or meaning about the words.

In recent years, following the trends of other areas like computer vision, neural information retrieval has gained enormous popularity according to Mitra and Craswell [42]. In a neural approach, the objective is to use a neural model to directly learn a specific task from the data (raw text). The idea is that the neural model will learn what features are the most important in order to perform the retrieval task.

The two most common tasks that use a neural model in IR are: *query-document matching* and *query-document ranking*, also known as learning to match and Learning to Rank (LTR), respectively.

3.2.1 Query-Document matching

According to Li and Xu [43], this problem consists in learning a matching function $f(x,y)$ that computes a similarity between two objects x and y from two different spaces X and Y, given the training data T composed of triples (x,y,r), where r is the relevance between x and y, i.e, its similarity. So the *query-document matching* problem, in terms of neural IR, tries to find a function $f(d,q)$, using neural networks, that computes the similarity between a document and a query, where the objective is to assign high similarity scores to query-document pairs that share some meaningful information.

This *learning to match* optimization problem is the same presented in Section 2.1, where the loss function most commonly used in the literature is the negative log likelihood of document being similar to a given query, as shown in Equation 3.6.

$$loss = -log\left(\prod_{(d^+,q,r=1)\in T} P(d^+|q)\right) \tag{3.6}$$

An important note regarding the previous equation is that the calculation of $P(d^+|q)$ requires computation of a probabilistic distribution over all the documents in a collection, as also shown in Equation 2.6, which is expensive, so an alternative is to select only a small subset of documents to approximate the probabilistic distribution.

According to Onal *et al.*, there are two types of neural architectures commonly used to implement the relevance matching function $f(d, q)$, namely *Representation Based* and *Interaction Based*, as shown in Figure 3.3. In a *Representation Based* architecture[1], left side in Figure 3.3, the query and the document are processed by a separate neural network, creating a separate representation for the query and the document, the matching is achieved by computing similarity over the two representations, e.g, using cosine similarity. In contrast, the *Interaction Based* architecture, uses a neural network to first create a joint representation of the query-document pair and then compute the similarity score over that representation.

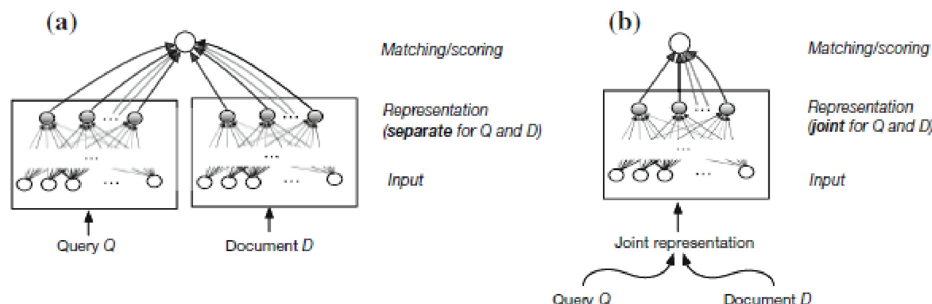


Figure 3.3: Two neural architectures used to implement the function $f(D, Q)$. In a) *Representation Based architecture*, b) *Interaction Based architecture*. The image was taken from [36].

3.2.2 Query-Document ranking

The Learning to Rank (LTR) problem consists in learning a function f, also known as ranking function, capable of ordering a set of objects. In terms of IR the LTR objective, according to Liu [44], is to produce a ranked list of documents given the relevance between these documents and the query, in a way that the most relevant documents should be ranked higher. A neural model can be the implementation of the function f, in this case, Liu [44] also categorizes three ways of training the neural model, i.e, defines three distinct loss functions as the training objective, **Pointwise**, **Pairwise** and **Listwise**.

- **Pointwise** approach assigns real number values as relevance score between each query-document pair and the objective is to predict the correct score for each query-document

[1] Some authors also use the term *Siamese network* to identify this neural architecture.

pair, $(q, d) \sim r$. The most used loss functions are the *cross-entropy* Equation 2.7 and *MSE* Equation 2.8.

- **Pairwise** approach assigns a relevance preference between two documents to a query and the objective is to give a higher score to the most relevant document in the pair. More formally, given the triple (q, d^+, d^-), if $r_{d^+} > r_{d^-} \Rightarrow F(q, d^+) > F(q, d^-)$, where $F(q, d)$ is the score computed by the neural model and r_d is the real relevance. In the literature the common loss function is the *hinge* [45] function, shown in Equation 3.7.

$$hinge = max(0, 1 - F(q, d^+) + F(q, d^-)) \tag{3.7}$$

  where d^+ denotes a document that should be ranked higher than d^-. Another popular loss function is the *RankNet Loss* [46] function Equation 3.8, which gives the probability that document $F(q, d^+)$ be ranked higher than document $F(q, d^-)$.

$$RankNet = \frac{1}{1 + e^{(-\sigma(F(q,d^+) - F(q,d^-))}} \tag{3.8}$$

  where σ denotes the sigmoid function.

- **Listwise** approach tries to directly optimize a ranking metric, e.g, Normalized Discounted Cumulative Gain (nDCG). Generally, this is more challenging since these metrics usually are not differentiable with respect to the model parameters. An example of a loss function is the *LambdaRank* [47] function, Equation 3.9, which uses the ΔnDCG to directly weight the gradients from the *RankNet* loss.

$$\nabla LambdaRank = \nabla RankNet \times |\Delta nDCG| \tag{3.9}$$

  So *LambdaRank* combines a pairwise loss (*RankNet*) with the nDCG metric, in such way that the *RankNet* gradients should linearly scale given the size of the change of the nDCG metric, i.e, by swapping the rank positions of two ranked documents, while leaving the others rank positions unchanged. A possible intuition is to visualize the *RankNet* gradients as directional vectors and the difference of nDCG as a weight that will change the vector magnitude and this way enforcing a signal from a non-differentiable metric. A more detailed analysis can be found in the following article [48]. Note that the *LambdaRank* loss does not have a single error value, instead it directly outputs the list of gradients that will be used to update the model.

3.2.3 Word Representation

As previously mentioned, a neural model is capable of using "raw" text as input in order to learn some specific task. However a neural model is a mathematical function that needs numerical data as the input, so this subsection is concerned with the conversion of raw text data into numerical data.

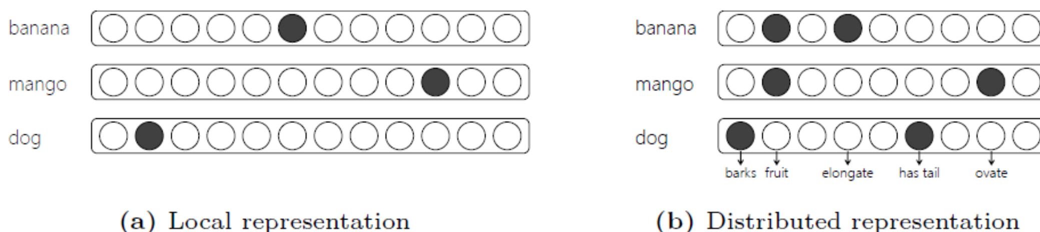
In IR the smallest unit of representation is considered to be the word (term)[2]. So a common solution is to encode the word into a numerical vector, which corresponds to the word representation. Mitra and Craswell [42] distinguish two types of word representation.

[2] Some times are the individual characters depending on the task.

- **Local representation**, also known as one-hot encoding, maps each word to a binary vector $\vec{v} \in 0, 1^{|V|}$, where each dimension corresponds to a word of the vocabulary V. This type of representation creates sparse vectors.
- **Distributed representation**, also known as embedding, maps each word to a fixed size real valued continuous vector $\vec{v} \in \mathbb{R}^{|K|}$, where the number of dimensions K usually ranges from 50 to 300 and each dimension encodes some property about that word. This type of representation creates dense vectors.

The distributed representation imposes a difficult challenge of decomposing a word in a set of fixed dimensions. Moved by the idea of *distributed hypothesis* [49], that states that words that occur in the same context tend to be similar, some algorithms try to explore this property and learn word representation directly from the data, like Word2Vec [50].

A visual representation of the local and distributed representation is presented in Figure 3.4. From the figure, it is clear how the distributed representation can encode the similarity between the words. For example, the word "banana" is more similar to the word "mango" when comparing with the word "dog", due to the fact that "banana" and "mango" share similar value along the "fruit" dimension[3]. On the other hand, it is not possible to extract this kind of information with the local representation, since all the vectors are orthogonal.



(a) Local representation

(b) Distributed representation

Figure 3.4: Visual illustration of the a) local representation and b) distributed representation. Circle represent each individual dimension and the painted circles represent a value, which in the case of local representation is 1. The image was taken from [42].

In the current literature, the most frequent used representation in neural models is distributed, since the local representation creates a considerable larger vector, which injuries the neural network training and performance. Besides that, the ability of the distributed representations to group similar words together helps the neural models generalize better for new data.

3.2.4 Word2Vec

The *word2vec* algorithm, introduced by Mikolov *et al.* [50], uses a shallow neural network to create a distributed representation of the words directly from big text collections, which makes it an unsupervised algorithm. The network idea is to represent words that appear in a similar context with similar vectors since according to the *distributed hypothesis* [49] these words will have a similar meaning.

In general, the neural network architecture, which is illustrated in Figure 3.5, is composed of an input layer of the size of the vocabulary V, one hidden layer with the size of our

[3]For sake of simplicity it was assumed that each dimension directly corresponds to a semantic dimension.

embedding vectors N and one output layer with the size of the vocabulary. The connections between the neural layers are represented by the weight matrices $\underset{V \times N}{W_{IN}}$ and $\underset{N \times V}{W_{OUT}}$, which corresponds to the learned word embeddings. So in more detail, this neural network produces two embedding vectors for each word w_i, the *in* vector represented as $\vec{v_{w_i}}$ and the *out* vector represented as $\vec{u_{w_i}}$.

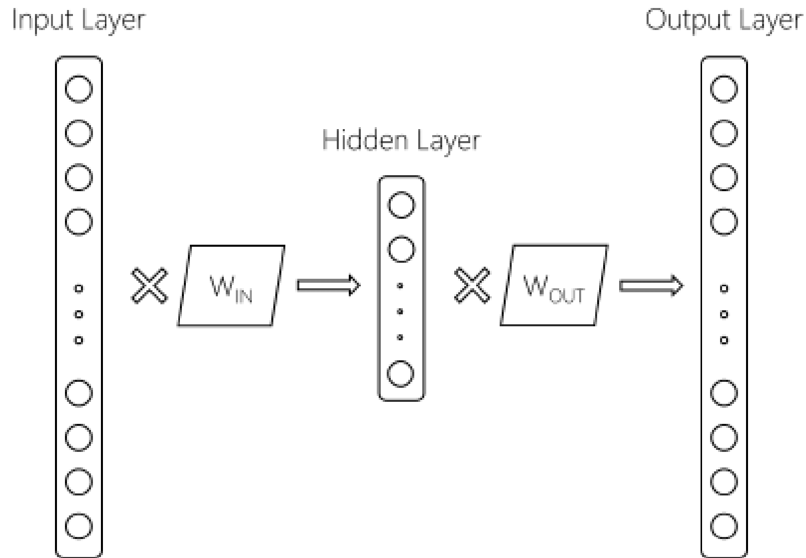


Figure 3.5: Overview of the word2vec neural network architecture. W_{in} and W_out are the weight matrices, previously introduced as θ, i.e, these matrices correspond to the connections between the neural units. The image was withdrawn from the article [51].

For the training process, the corpus is firstly scanned using a fixed size window, which produces the training samples. The window is composed of a central word and their neighbour words also called context words. Formally, given i as the index of the *i-th* word of the training corpus and c as half of the window size. A window W of size 5 ($c = \frac{5}{2} \approx 2$) is represented as $W = \{w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}\}$, where a central word is represented as w_i and the context words as $w_{i \pm c} = \{w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}\}$. Mikolov *et al.* proposed two approaches for the optimization problem, the continuous Skip-Gram (SG) model and Continuous Bag of Words (CBoW) model, illustrated in Figure 3.6,

The SG consists in trying to predict the context words, given the central word of the window. So, given a sentence of size T, the training objective is to minimize the negative average log likelihood of $w_{i \pm c}$ being the context words of w_i. Equation 3.10 shows the mathematical formulation.

$$loss = -\frac{1}{T} \sum_{t=1}^{T} \sum_{j=-c, j \neq 0}^{c} log\left(P(w_{i+j}|w_i)\right) \tag{3.10}$$

In contrast, the CBoW tries to predict the central word, given the context words of the window. So, the training objective is to minimize the negative average log likelihood of w_i being the central word given the context words $w_{i \pm c}$. Equation 3.11 defines a mathematical

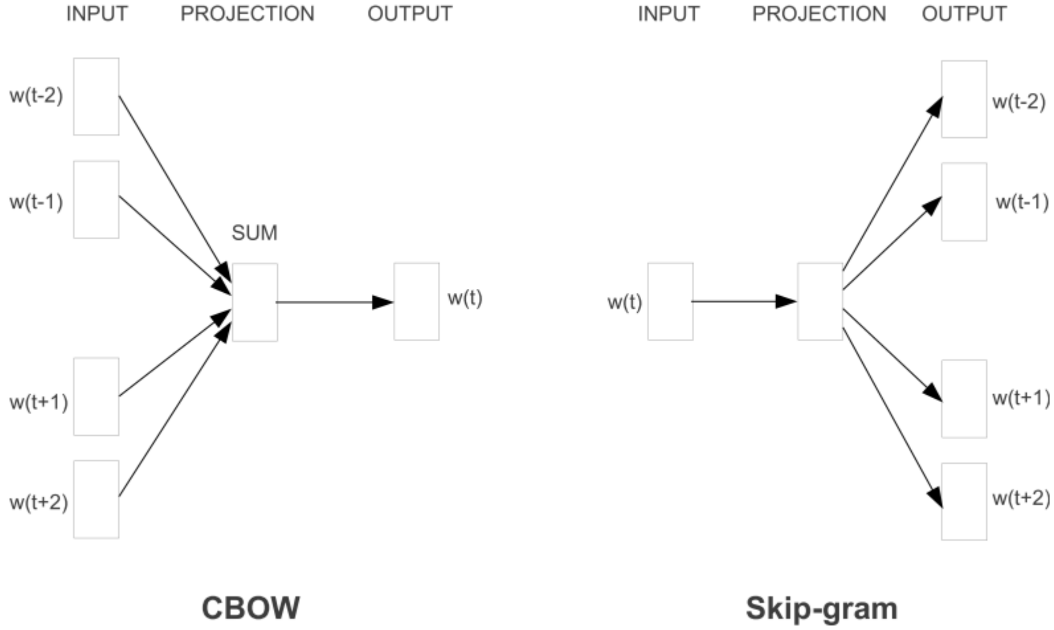


Figure 3.6: Overview of the Skip-Gram and Continuous Bag of Words approach. The image was withdrawn from the article [50].

formulation and e_c corresponds to the summation of the embeddings of the context words $w_{i \pm c}$.

$$loss = -\frac{1}{T}\sum_{t=1}^{T} log\left(P(w_i|e_c)\right) \tag{3.11}$$

In both Equations 3.10 and 3.11, the conditional probability is computed by the softmax operation that is shown in Equation 3.12. This equation was derived from the generic softmax equation previously presented in Equation 2.6. For that, the set of discrete classes (C) corresponds to the words of the vocabulary V and the network score for each class (word) w_C given the input word w_I (Z_{w_C,w_I}) corresponds to the dot product between the *out* vector of w_C and the *in* vector of w_I, i.e, $Z_{w_C,w_I} = \left(u_{\vec{w_C}}{}^{T} \cdot v_{\vec{w_I}}\right)$

$$\begin{aligned} P\left(w_C|w_I\right) &= \frac{e^{\left(Z_{w_C,w_I}\right)}}{\sum_{j=1}^{|V|} e^{\left(Z_{w_j,w_I}\right)}} \\ P\left(w_C|w_I\right) &= \frac{e^{\left(u_{\vec{w_C}}{}^{T} \cdot v_{\vec{w_I}}\right)}}{\sum_{j=1}^{|V|} e^{\left(u_{\vec{w_j}}{}^{T} \cdot v_{\vec{w_I}}\right)}} \end{aligned} \tag{3.12}$$

It was also shown that softmax is a computationally expensive operation when the number of discrete classes is high, which is the case. To address this issue Mikolov *et al.* [52] show two approximations, the *hierarchical softmax* and the Negative Sampling (NEG). The *hierarchical softmax*, first introduced by Morin and Bengio [53], uses a tree structure to distribute the V discrete classes, which reduces the computation complexity of the softmax to $log(V)$. On the other hand, NEG is a simplification of a well known Noise Contrastive Estimation (NCE), introduced by Gutmann and Hyvärinen [54], which states that model should be able to

differentiate data from noise by means of logistic regression, i.e., NCE transforms the multi-class problem to a binary classification problem. Furthermore, word2vec is only concerned with the quality of the embedding vectors and not with the quality of the output probabilistic distribution. This allowed to simplify NCE resulting in the NEG Equation 3.13.

$$\begin{gathered}
log(P(w_C|w_I)) \approx \underbrace{log\left(\sigma\left(Z_{w_C,w_I}\right)\right)}_{\text{Positive sample}} + \sum_{j=1}^{k}\left[\mathbb{E}_{w_j \sim P_n(V)} \underbrace{log\left(\sigma\left(-Z_{w_j,w_I}\right)\right)}_{\text{Negative sample}}\right] \\
log(P(w_C|w_I)) \approx \underbrace{log\left(\sigma\left(\vec{u_{w_C}}^T \cdot \vec{v_{w_I}}\right)\right)}_{\text{Positive sample}} + \sum_{j=1}^{k}\left[\mathbb{E}_{w_j \sim P_n(V)} \underbrace{log\left(\sigma\left(-\vec{u_{w_j}}^T \cdot \vec{v_{w_I}}\right)\right)}_{\text{Negative sample}}\right]
\end{gathered} \tag{3.13}$$

In NEG a sample is positive if the context words are neighbour of the central word otherwise, the sample is considered to be negative. So, this formulation uses k negative samples for each positive sample and the negative samples are randomly selected by the probabilistic distribution $P_n(V)$. The idea is that the model will only update the weights associated with the k words of the negatives samples and with the word of the positive sample. In the experiments of Mikolov *et al.*, the value of k between $5 - 20$ produces good results for small corpora and $2 - 5$ can be used in bigger corpora. For the probabilistic distribution, $P_n(V)$ they used an unigram distribution $U(w_i)$ raised to the power of $\frac{3}{4}$[4], Equation 3.14.

$$U(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=1}^{|V|} f(w_j)^{\frac{3}{4}}} \tag{3.14}$$

Here, the function $f(w_i)$ corresponds to the frequency of the word w_i in the corpus.

3.2.5 Word2Vec subword extension

The *word2vec* algorithm is only capable of learning a vector representation (embedding) for words that are present in the training data. As a consequence, it is impossible to get an embedding vector for a never seen word, which makes it difficult to reuse the learned vectors. This problem, in the literature, is known as an Out-of-Vocabulary (OOV) entry.

Bojanowski *et al.* [55], proposed an extension to the continuous Skip-Gram (SG) model that alleviates this problem, by incorporating subword information (word morphology) in the training process. For that, the model learns a vector representation (embedding) for each subword unit and the word embedding is given by the sum of its subword embeddings. So, a word is defined as a bag of character n-gram and by the word itself, here the n-gram corresponds to a subword unit and note that n can belong to an interval. For example, using $n \in \{3, 4, 5\}$ (3-gram to 5-gram), the word *"banana"* is represented by the following bag of characters $\{< ba, ban, ana, nan, na >, < ban, bana, anan, nana, ana >, < bana, banan, anana, nana >, < banana >\}$.

[4]This value was found to produce good results

More formally, considering $G(w_I)$ as a set that contains all the n-gram of the word w_I, the network score for each word w_C given the input word w_I (Z_{w_C,w_I}) of this SG extension is given by the Equation 3.15.

$$Z_{w_C,w_I} = \sum_{g \in G(w_I)} \vec{v_{w_g}}^T \cdot \vec{u_{w_C}} \tag{3.15}$$

This formulation can directly replace the original network score function in the Equations 3.12 and 3.13. Note that, the embedding of the i-th word (w_i) is given by $\vec{v_{w_i}} = \sum_{g \in G(w_I)} \vec{v_{w_g}}$. Additionally, when comparing this score function to the original, it becomes more clear that the sum of the n-gram embeddings directly replaces the *in* vector of the word w_i, $\sum_{g \in G(w_I)} \vec{v_{w_g}} \rightarrow \vec{v_{w_I}}$. Which explains why this extension produces better representations for rare words since a rare word is composed by a set of n-grams that are also shared with other words, thus a n-gram is more likely to have a greater frequency when compared to the rare word.

For the optimization process, the authors also used the NEG formulation with 5 negatives samples and n-gram between 3 and 6. In terms of results, the authors observed that the learned embeddings outperform the original SG model and also produce robust embeddings for OOV words. Finally, this extension was open-sourced in the *fasttext* library[5].

3.3 Evaluation Metrics

In IR, the developed systems are compared using a set of metrics that measures their effectiveness. This section will address the metrics that are used over this dissertation, but a more complete overview is available in the following book [56].

To evaluate a IR system let us first define the *gold standard* as a set of relevant documents to a given query $G = \{d_1^+, d_2^+, ..., d_n^+\}$, $R_m = \{d_1, d_2, ..., d_m\}$ as the ordered set of top m documents retrieved by the system to the same query and Q as a set of queries to be tested by the system.

Recall, Equation 3.16, gives the probability of a relevant document being retrieved by the system, i.e, recall is the fraction of relevant documents that are retrieved.

$$Recall = \frac{|G| \cap |R_m|}{|G|} \tag{3.16}$$

Precision, Equation 3.17, gives the probability of a retrieved document being relevant, i.e, recall is the fraction of retrieved documents by the system that are relevant.

$$Precision = \frac{|G| \cap |R_m|}{|R_m|} \tag{3.17}$$

Additionally, both recall and precision can be computed at a *cut-of rank* of K, which means that the metric is only performed to the top K retrieved documents R_k, instead of all the retrieved documents R_m. In this situation, the metrics are designated of **Recall@**K and **Precision@**K.

[5] https://fasttext.cc/

F_1 **score**, Equation 3.18, is a measure that performs a weighted harmonic mean of the precision and recall.

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{3.18}$$

Mean Average Precision (MAP), Equation 3.19, is a mean of the **Average Precision**, that computes the precision in each rank, this will approximate the area under the precision-recall curve [56].

$$MAP = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \underbrace{\frac{1}{|G_q \cap R_m|} \sum_{k=1}^{|R_m|} Precision@k \times rel(k)}_{\text{Average Precision}} \tag{3.19}$$

Here, $rel(k)$ is a binary function that indicates if the retrieved document at position k is relevant.

Mean Reciprocal Rank (MRR), Equation 3.20, is the average of the reciprocal ($\frac{1}{x}$) rank of the first relevant result, retrieved by the system.

$$MRR = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{rank_1(R_m)} \tag{3.20}$$

Here, $rank_1$ is a function that returns the position of the first retrieved document that is relevant.

CHAPTER 4

State of the Art

This chapter presents the state of the art in terms of neural models for IR. It is divided into two sections, that address the query-document matching and learning to rank tasks. Some of the state of the art models are also previous submissions to the BioASQ competition.

4.1 Query-Document Matching

In this subsection are presented state of the art neural solutions to the query-document matching task, previously introduced in Section 3.2.1. In general, the neural model will learn how to project the queries and the documents to the same n-dimensional space. Ideally, in this space, the relevant documents and the query will be grouped in terms of similarity. Then the closest documents could be retrieved for a specific query.

4.1.1 Average word embedding

Average Word Embedding (AWE) is a simple technique to create a dense vector representation of sentences over an embedding space. In a neural IR perspective, the queries and the documents are represented by the average of their embedding words, i.e, the queries and the documents are projected to an n-dimensional embedding space. So, for a given text t, that can be a query or a document, its dense vector representation, $\vec{t}$, is obtained following Equation 4.1. Then the relevant documents to a query are retrieved by the cosine similarity, Equation 3.1.

$$\vec{t} = \frac{1}{|t|} \sum_{i=1}^{|t|} \frac{\vec{w_i}}{\|\vec{w_i}\|} \tag{4.1}$$

Here, $\vec{t}$ correspond to the vector representation of the text t and $\vec{w_i}$ is the embedding vector of the i-th word of the text t. Note that $\frac{\vec{w_i}}{\|\vec{w_i}\|}$ corresponds to a normalization step of the embedding vector $\vec{w_i}$ to a unit vector.

Kosmopoulos *et al.* [57], improve the previous idea by combining it with the TF-IDF weights, i.e, the AWE becomes a weighted arithmetic mean as shown in Equation 4.2. They

apply this technique to obtain low dimensional text representation to feed an hierarchical text classification algorithm, for automatically assigning MeSH labels to text.

$$\vec{t} = \frac{\sum_{i=1}^{|V|} \frac{\vec{w_i}}{\|\vec{w_i}\|} \times tfidf_{w_i,t}}{\sum_{i=1}^{|V|} tfidf_{w_i,t}} \tag{4.2}$$

AWE in document retrieval

Years later, Brokos *et al.* [58] used this technique in a document retrieval task applied to the biomedical domain. In more detail, they compared three variants of this technique with the PubMed search engine. The four systems compared were:

- Cent - Correspond to the normal AWE, Equation 4.1.
- Cent-IDF - Correspond to the weighted arithmetic mean, Equation 4.2.
- Cent-IDF-RWMD - Same as the previous, but Word Mover Distance (WMD) was used to rerank the results.
- PubmedSE - Pubmed Search Engine.

Also in their experiments, they used 1307 biomedical queries from BioASQ challenge as test-set and documents were from the Pubmed collection, which at that time contained approximately 14 million articles. In terms of results, *Cent-IDF* had similar performance to the *pubmedSE* for the MAP and Mean Interpolated Precision (MIP) metrics, which is surprising since *Cent-IDF* is a much simpler and fast technique when compared to the highly engineered system behind *PubmedSE*. As expected the *Cent-IDF-RWMD* got the best results.

The authors also report that *PubmedSE* failed to retrieve documents for 35% of the queries, which indicates a limitation for retrieval systems based on keyword match.

AWE in snippet retrieval

In the work of Galkó and Eickhoff [59], they applied the previous techniques, Equation 4.1 and 4.2, to a task of snippets retrieval. The objective in this type of task is to retrieve short text passages, snippets, from documents given a specific query. The results were compared, in terms of MAP, precision, recall, and F_1, against some other neural models like the state-of-the-art Deep Relevance Matching Model (DRMM).

In their experiment, they used the 2017 BioASQ data, with 1799 biomedical questions and test set with 500 biomedical questions. For each given question the relevant documents were split into individual sentences, i.e, snippets. Then the retrieval mechanism will score the resulting set of snippets.

Both techniques, AWE and weighted AWE, were capable of reaching a similar performance to the DRMM model. These results are highly influenced by the small amount of training data, but as the authors said, this is a fast and non-trainable[1] method capable of achieving good results, especially with small datasets.

[1]The process of learning the embedding is not considered

4.1.2 Dual Embedding Space Model

Moved by the idea that the *aboutness* of a document can be captured by the relationship between the query terms and all the terms in the document, Mitra *et al.* [51] proposed the Dual Embedding Space Model (DESM). This model utilizes two different embedding spaces for the queries and documents projections, which shows to be able to capture more topical relation between the query and the document words. For example, given the query *Alburquerque*, two passages were retrieved, as shown in Figure 4.1. Both passages contain the term *Alburquerque*, in yellow, but clearly, the first passage (a) is much more relevant to the query that the second passage (b). Due to the fact that the passage (a) contains more topical similar terms, to the query term *Alburquerque* represented in green like *population* and *metropolitan*. In other words, the passage (b) merely refers the *Alburquerque*, while the passage (a) is about the *Alburquerque* city.

Albuquerque is the most populous city in the U.S. state of New Mexico. The high-altitude city serves as the county seat of Bernalillo County, and it is situated in the central part of the state, straddling the Rio Grande. The city population is 557,169 as of the July 1, 2014, population estimate from the United States Census Bureau, and ranks as the 32nd-largest city in the U.S. The Metropolitan Statistical Area (or MSA) has a population of 902,797 according to the United States Census Bureau's most recently available estimate for July 1, 2013.

(a)

Allen suggested that they could program a BASIC interpreter for the device; after a call from Gates claiming to have a working interpreter, MITS requested a demonstration. Since they didn't actually have one, Allen worked on a simulator for the Altair while Gates developed the interpreter. Although they developed the interpreter on a simulator and not the actual device, the interpreter worked flawlessly when they demonstrated the interpreter to MITS in Albuquerque, New Mexico in March 1975; MITS agreed to distribute it, marketing it as Altair BASIC.

(b)

Figure 4.1: Two examples of passages retrieved for the query *Alburquerque*. In yellow, is represented the exact match and in green are the topical similar words. The image was withdrawn from the article [51].

The authors decided to use the two embeddings spaces that are learned during the *word2vec* training process since the CBoW and SG training objective effectively captures the words co-occurrence. As previously shown in 3.2.4, the *word2vec* learn two embedding vectors for each word, that correspond to the *IN* and *OUT* projections. When the training finishes only one space, usually the *IN* space is kept. However, the authors were able to observe that the similarity between the two spaces, *IN* and *OUT*, encoded a more topical relationship between words. This property can be verified in Table 4.1, that compares the most similar words to the query term *eminem*, when using the different embedding spaces. *IN-IN* and *OUT-OUT*, correspond to the projection of the query term to the, *IN* and *OUT* embedding space respectively. On the other hand, *IN-OUT* corresponds to the projection of the query term using the *IN* embedding to the *OUT* embedding space. From the table is visible that the *IN-IN* and *OUT-OUT* retrieve similar words and there are semantically equivalents, in this case, other artist names. But in the case of the *IN-OUT*, the most similar words are words that give an idea of *aboutness* about the query term, like *rap*, *featuring* and *tracklist*. Once more, the cosine similarity is used to retrieve the most similar words to a term in an embedding space.

Mathematically speaking, the authors formulated the DESM model as an average of the cosine similarity between all the embedding query terms and the document embedding vector,

Table 4.1: Most similar words to "eminem", using different embedding spaces. The data was withdrawn from the article [51].

IN-IN	OUT-OUT	IN-OUT
eminem	eminem	eminem
rihanna	rihanna	rap
ladacris	dre	featuring
kanye	kanye	tracklist
beyounce	beyounce	diss

as shown in Equation 4.3.

$$DESM(q, D) = \frac{1}{|q|} \sum_{i=1}^{|q|} cos(\vec{q_i}, \vec{D}) \tag{4.3}$$

Here, $\vec{q_i}$ corresponds to the *IN* embedding of the i-th word belonging to the query q. $\vec{D}$ correspond to the AWE of the document terms using OUT space, Equation 4.1. The function *cos* is the cosine similarity previously shown in Equation 3.1.

In their experiments, the authors found that the DESM was prone to retrieve false positives, but at the same time was capable of model the important aspect of document *aboutness*. So the authors proposed a mixed model, where the BM25 ranking function is combined in a linear fashion with the DESM model, as demonstrated in Equation 4.4. BM25 was an interesting choice because it is a ranking function based on the exact match between query and document terms and commonly misses semantically or topical similar documents which is the strong point of the DESM model.

$$MM(q, D) = \alpha DESM(q, D) \times (1 - \alpha) BM25(q, D) \tag{4.4}$$

Where α is a hyperparameter that will control the weight given to each model. Considering that the BM25 produces absolute ranking scores, we can say that in this linear combination, the DESM model will rearrange the final ranking scores of the BM25, in order to "push" the score of documents that are more topical related to the query, i.e, the DESM model will act as a weaker ranker over the BM25.

In term of the evaluation, the authors compare the performance of the following models, BM25, Latent Semantic Analysis (LSA), DESM and Mixed model (DESM+BM25) using the nDCG metric. Overall the Mixed model got better results, which reinforces the idea that the DESM model gives a positive boost in performance to the BM25.

4.1.3 Deep Semantic Similarity Model

An alternative to the embedding representation of the queries and documents is to use a DNN to directly learn a dense vector representation of the queries and documents from a set of relevant query-document pairs. The Deep Semantic Similarity Model (DSSM) or Deep Structured Semantic Model (DSSM) proposed by Huang *et al.* [60], was one of the pioneering models, that used DNN to learn a semantically similar low-dimensional space for

the queries and documents. The model follows a *Representation Based* architecture, where two neural models are used to separately learn a low-dimensional representation of the query and documents. The neural models are composed of a fully connected layers and the final score between the two models is obtained through cosine similarity.

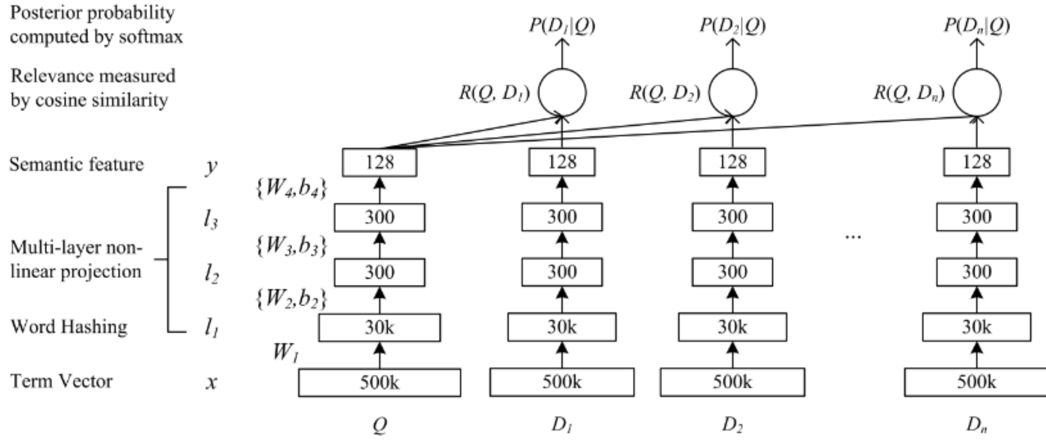


Figure 4.2: . The image was withdrawn from the article [60].

A more detailed view of the architecture is given in Figure 4.2. The input text (documents or queries) is represented as Bag of Words (BoW), which correspond to the input layer of the network. After that, the authors introduced the *word hashing* layer, that is a non-trainable layer, with the objective to reduce the input dimensionality. Then, they used three fully connected layers, where the first two are hidden layers with 300 units and the last one is the output layer with 128 units. The *hyperbolic tangent* was the activation function used. Finally, the DSSM model computes a score between the two neural models outputs using the cosine similarity.

In terms of the learning process, the objective is to maximize the conditional likelihood $P(D|Q)$ of a relevant document to a given query. Once more, the conditional probability is computed using the softmax operation over all the documents scores, Equation 2.6, since the number of documents tends to be large, they use a technique similar to NEG to approximate the probabilistic distribution. So for each positive pair (query - relevant document), the authors randomly select four negative pairs (query - non-relevant document) and the probabilistic distribution will only be computed over this total of five pairs. The loss function that will be minimized is the negative log likelihood, Equation 3.6.

The authors used click-through data logged by a commercial web search engine, where a positive pair is composed by the user-query and the respective clicked document. They split the data into training and validation set and compared the results against other models/techniques using the validation set. The evaluation metric was the nDCG and the DSSM model had the best results when comparing to the *TF-IDF*, *BM25*, *LSA* and *Deep auto-encoder (DAE)* [61].

Word Hashing Layer

This layer aims to reduce the dimensionality of the Bag of Words (BoW) vectors, used to represent the queries and the documents. The method consists in transforming a word into

letter n-gram. For example, the word *good* can be represented by the following letter trigrams $\{\#go, goo, ood, od\#\}$, where # is an initial and final word delimitor.

On the negative side, this technique produce *collisions* between words, i.e, two different words can have the same letter n-gram representation. The authors reported that a vocabulary with 500k words can be represented by a 30621 letter trigram vector (approx. 16 times smaller) with only 22 cases of collisions, which are negligible.

Since the number of letter n-gram tends to be finite, this technique is more robust to the OOV problem, because it is possible that the decomposition of a never seen word can be mapped by the preexistence letter n-gram vector. A common example are the composed words since they are morphologically similar.

DSSM Variants

This model opened the doors to a new strand of DSSM models, based on bridging its weakness. The most popular variants are the following.

- **Convolutional Latent Semantic Model (CLSM)** [62] is an extension of the DSSM, that uses convolution and max-pooling layers instead of the fully connected layers. In this model the text is represented as a set of words $t = \{w_1, w_2, ..., w_n\}$ and each word is encoded as bag-of-trigrams. Then a convolution layer will extract local contextual features from fixed sized window (*filter*) of words, then a max-pooling layer will only capture the most relevant ones since the meaning of a sentence is determined by few words.
- **LSTM Deep Structured Semantic Model (LSTM-DSSM)** [63] is also an extension of the DSSM, that use LSTM layers instead of the fully connected layer. Since the LSTM is capable of handle sequential data, the input text is also represented as a set of sequential words $t = \{w_1, w_2, ..., w_n\}$, encoded as bag-of-trigrams.

Both variants use the same optimization objective of the DSSM model. Palangi *et al.* [63] used click-through data logged by a commercial web search engine, to train the three models and they reported that the LSTM-DSSM model outperforms the DSSM and CLSM model.

4.1.4 Word Mover Distance

So far, the similarity between sentence was calculated by the cosine similarity between a dense vector representation of the sentences. The Word Mover Distance (WMD), proposed by Kusner *et al.* [64], describes a distance function[2] between two sentences. More specifically, this function measures the cost needed to transform the first sentence into the second one over the embedding space. Small values of the cost mean these sentences are semantically similar.

More formally, WMD is the minimum cumulative distance that words from sentence A, in an embedding space, need to travel to be able to exactly match the words of the sentence B, in the same embedding space. The distance between words in the embedding space is computed using the Euclidean distance. WMD can be seen as a transportation optimization problem,

[2]Note that distance function is equal to $1-$similarity function

defined in Equation 4.5, and more precisely is a special case of the well-studied Earth Mover Distance (EMD) [65].

$$
\begin{aligned}
wmd(s_a, s_b) = \min_{T \geqslant 0} & \sum_{i,j=1}^{|V|} T_{i,j} \times \|\vec{w_i} - \vec{w_j}\| \\
\text{Subject to:} & \sum_{j=1}^{|V|} T_{i,j} = d_i \\
& \sum_{i=1}^{|V|} T_{i,j} = d_j \\
\text{Where:} \; & V = s_a \cup s_b
\end{aligned}
\tag{4.5}
$$

Here, each entry of the flow matrix $T \in \mathbb{R}^{|V| \times |V|}$ represents how much of word i from the sentence a travels to the word j from the sentence b. d_i and d_j are distributions of words in the sentences a and b, respectively. The two restrictions ensure that sentence a is completely transformed in sentence b. Finally, $\|\vec{w_i} - \vec{w_j}\|_2$ corresponds to the Euclidean distance between the embedding vectors of the word i and j from the sentences a and b.

To better understand the intuition behind WMD lets consider the example presented in Figure 4.3. First the non-relevant words are removed resulting in the following sentences (a) *Obama speaks media Illinois* and (b) *President greets press Chicago*. Then the objective is to find a matrix T capable of minimizing the Equation 4.5, where the distribution of all the words in a and b is $\frac{1}{4}$, i.e, $d_i = \frac{1}{4}$ $\forall i \in 1, ..., |a|$ and $d_j = \frac{1}{4}$ $\forall j \in 1, ..., |b|$. The final result of the matrix T following the example must be, $T_{Obama,President} = \frac{1}{4}$, $T_{speaks,greets} = \frac{1}{4}$, $T_{Illinois,Chicago} = \frac{1}{4}$ and $T_{media,press} = \frac{1}{4}$, where the other entries are equal to 0. This was a trivial case because the sentences have the same number of words. When this does not happen normally one word is transported to multiple words.

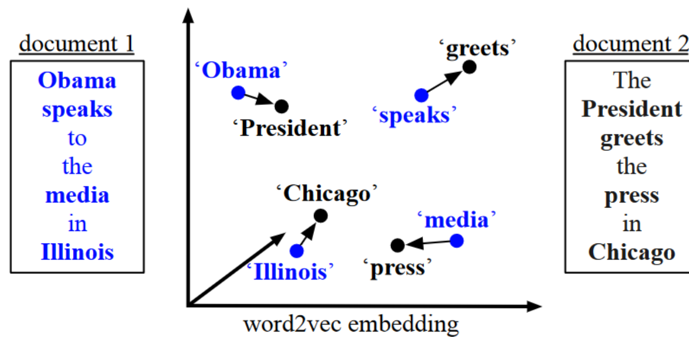


Figure 4.3: . The image was withdrawn from the article [64].

Regardless of whether there are common words between the sentences, the WMD will always return small distances for semantic similar sentences, since in the embedding space, similar words tend to be closer to each other.

Unfortunately, the computation complexity of this method is $O(|V|^3 log(|V|))$, which directly scales with the number of unique words (vocabulary |V|) in both sentences. In

order to improve this complexity, the authors shown a lower-bounder approximation to the WMD, which is Relaxed Word Moving Distance (RWMD), in this relaxation one of the two restrictions in Equation 4.5 is forgotten.

The authors reported superiors results of the WMD performance in eight datasets, when compared with some baselines like TF-IDF, BM25.

Query-Document Word Mover Distance

The original WMD formulation is not suitable for the *query-document* match problem since the two sentences to be compared, query and document, are a lot different. Usually, a query consists of a small number of words, aiming for a piece of specific information. On the contrary, the document contains a large number of words and diverse information.

Kim *et al.* [66] propose a new version of the WMD applied to the *query-document* matching problem. To tackle the previously stated problems the authors made the following changes:

- The embedding space was trained in the documents collection.
- Introduced the Inverse Document Frequency (IDF) as weight in the distribution of the words d_i, moved by the idea that more important words should have higher weight and in turn bigger distribution.
- Instead of the Euclidean distance, the cosine similarity is used.
- The minimization became a maximization, i.e, $\max_{T \geqslant 0} \sum_{i,j=1}^{|V|} T_{i,j} \times cos(\vec{w_i}, \vec{w_j})$, since the dissimilarity measure (Euclidean distance) change to a similarity measure (cosine similarity).

4.2 Learning to Rank

In this subsection are presented state of the art neural solutions to the query-document ranking problem. In general, the neural model will learn the interactions between a query and the documents giving a real number score value, that can be used to retrieve the most relevant documents.

4.2.1 Deep Relevance Matching Model

The previous models/techniques were based on a *Representation Based* architecture, where the queries and documents representation were created/learned separately. On the other hand, the Deep Relevance Matching Model (DRMM) proposed by Guo *et al.* [67], explores the *Interaction Based* architecture, where a query and document representation is learned together.

The author's design a deep neural model focused in relevance matching, instead of the semantic matching, since in their perspective the semantic matching objective is not suitable for the ad-hoc retrieval task, especially of large documents. In their perspective the most important factors for relevance match are:

- **Exact matching signals:** The exact match between document and queries terms is the most important signal in ad-hoc retrieval, which explains the performance of more traditional models like BM25.

- **Query term importance:** The queries tend to be short and simple, most commonly based on keywords. So the importance of each query term is an important feature to take into account.
- **Diverse matching requirement:** The documents are usually long and relevant information can be only a small subset of a greater document. So a relevance matching could happen in any part of a document.

So, based on these three factors the DRMM model was created, using a **matching histogram mapping**, a **feed forward matching network** and a **Term gating network**, as show in Figure 4.4.

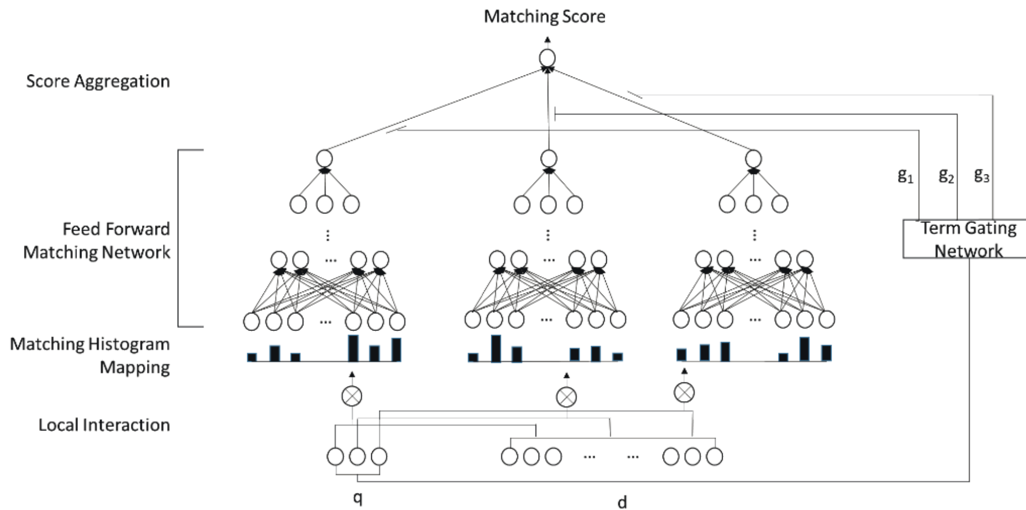


Figure 4.4: Architecture of the Deep Relevance Matching Model. The image was withdrawn from the article [67].

The **matching histogram mapping** is concerned into capture the interactions of each query term and all the documents terms, since the length of the queries and documents are variable the authors decided to represent each interaction between the query term and the document with histograms. So the input of the model will be a set of histograms with the size of the query. Mathematically speaking this operation corresponds to Equation 4.6.

$$h(w_i \otimes D),\ \forall_i \in 1, ..., |Q| \tag{4.6}$$

Here, h corresponds to the histogram mapping function, $\otimes$ denotes the interaction operator[3] between a query term $w_i^{(q)}$ and all the documents terms D.

Equation 4.7 is derived from 4.6 and describes the practical implementation used by the authors to capture the word interactions. In it, they used the word embedding of size E, created from word2vec algorithm, to represent each word and therefore used the cosine similarity as the interaction operator $\otimes$. For the histogram h they used a set of 30 bins, which mean that the cosine output interval $[-1, 1]$ was subdivided into 30 discrete intervals. Then a

[3] In here as a different meaning of the previously used in Equations 2.11 and 2.12.

logarithm operation was applied to each bin of the histogram, with the intuition of ease the learning of multiplicative relationships.

$$log(h_{30}(cos(\underset{|D|\times E}{D}, \underset{E\times 1}{\vec{w_i^{(q)}}}))),\ \forall_i \in 1, ..., |Q| \tag{4.7}$$

To better understand the histogram mapping let us considered that the query-document interactions are captured in a histogram with 4 bins h_4. So, as previously seen, the histogram subdivide the cosine output into 4 intervals, $[-1, -0.5[, [-0.5, 0[, [0, 0.5[, [0.5, 1]$. Given a query $q = \{car\}$, a document $D = \{car, rent, truck, bump, injunction, runway\}$ and the cosine similarity between the query and document terms $cos(\underset{|D|\times E}{D}, \underset{E\times 1}{\vec{w_i}}) = \{1, 0.2, 0.7, 0.3, -0.1, 0.1\}$. It will result in the following histogram $h_4(\{1, 0.2, 0.7, 0.3, -0.1, 0.1\}) = [0, 1, 3, 2]$. Then, the final step is to apply logarithm to each bin, so $log(h_4) = [-inf, 0, 0.47, 0.3]$.

A **feed forward neural network** will receive the histograms as input and for each, will output a single result. So the network is composed of an input layer with 30 nodes, one node for each histogram dimension. Follow by a hidden layer with 5 nodes and an output layer with 1 node. All the layers apply the *hyperbolic tangent* as activation function. Note that this network will run over each individual histogram, so the number of histograms can be a variable parameter, which allows queries with different lengths. The authors also explain that more advanced neural networks, like CNN and RNN, will not improve the performance since the network input is positional independent (histogram dimension).

The **term gating network** will weight and aggregate each output of the previous network, i.e, this network will capture the individual importance of the query terms, which is related to the second factor in relevance matching **Query term importance**. For that, each query term $w_i^{(q)}$ is associated with a gate g_i that corresponds to the word importance. The gate g_i is computed by the softmax function Equation 4.8, where the resulting probabilistic distributing over all the query terms measure the individual importance of each term.

$$g_i = \frac{e^{(w_g \cdot x_i^{(q)})}}{\sum_{j=1}^{M} e^{(w_g \cdot x_j^{(q)})}} \tag{4.8}$$

In Equation 4.8 the weight w_g, denotes the trainable weight of the term gating network and $x_i^{(q)}$ denotes numerical representation of the i-th query term. The authors find out that the IDF as query term representation gives the best results, $x_i^{(q)} = idf_t$, so in this case the term gating network is reduced to just one trainable weight. Also note that this equation is a derivation of the more generic version presented in Equation 2.6 and the resulting conditional probability correspond to the term importance distribution.

The final DRMM output is obtained by a linear combination between each term gate and output of the feed forward network, Equation 4.9,

$$s = \sum_{i=1}^{M} g_i z_i \tag{4.9}$$

where M denotes the size of the query and z_i the output of the feed forward network for the interactions between the i-th query term and the document.

The authors trained the term gating network and the feed forward network in a end-to-end fashion, using a pairwise loss function, namely the *hinge loss* previously presented in 3.2.2. It is worth to mention, that this model is not capable of train the term embedding in an end-to-end fashion since the gradients can not be back-propagated through the histogram.

In their experiments the authors shown that this model outperforms strong baselines, like BM25, representation-focused models, like DSSM and Convolutional Deep Structured Semantic Model (C-DSSM). The results supported the author's hypothesis.

DRMM variants

Similar to the DSSM, the DRMM also was a pioneer model that open the doors to a new trend of relevance matching models that explore its weakness. More precisely, the histograms are not differentiable which makes it impossible to propagate the gradients to the embedding matrix. Besides that, the histograms also ignore the context in which the terms occur.

- **Attention Based ELement-wise DRMM (ABEL-DRMM)** [68] propose a differentiable *context-aware* encoding of the query or document terms, in order to capture their context in which the terms occur. The encoding is performed by a residual neural layer over the term t_i and their neighbours t_{i-1}, t_{i+1}, Equation 4.10 shows the layer operation.

$$c(t_i) = \gamma\left(\theta \cdot \underbrace{[\vec{t_{i-1}}; \vec{t_i}; \vec{t_{i+1}}]}_{concatenation} + b\right) + \underbrace{\vec{t_i}}_{residual} \tag{4.10}$$

Here, $c(t_i)$ represents the *context-aware* encoding of the i-th term t_i and $\vec{t_i}$ is its embedding representation. Then in order to capture the interactions between each encoded query term ($c(q_i)$) and all the document terms ($C(d_j) = [c(d_1); ...; c(d_j)]$), the model uses a attention mechanism over all the documents terms, instead of using the histogram alternative proposed in the original DRMM since it is non-differentiable.

$$\begin{aligned} a_{i,j} &= softmax\left(\underset{D\times E}{C(D)} \cdot \underset{E\times 1}{c(q_i)}\right) \\ d_{q_i} &= \sum_j a_{i,j} \times c(d_j) \\ \phi_H(q_i) &= \frac{d_{q_i}}{\|d_{q_i}\|} \otimes \frac{c(q_i)}{\|c(q_i)\|} \end{aligned} \tag{4.11}$$

Equation 4.11 describes the process to compute the interactions between the documents terms and the i-th query term ($\phi_H(q_i)$), or as they call *Doc-Aware Query Term*. That is given by an element-wise ($\otimes$) multiplication between an attention based representation of the document given the query term (d_{q_i}). This attention-based representation is given by a linear combination between the attention weights and the encoding of the document terms ($c(d_j)$), where the weights $a_{i,j}$ of the attention model are given by

a probabilistic distribution of the i-th query term with all the documents term, this distribution is computed by the softmax operation. Alternatively, Figure 4.5 presents a visualization of the ABEL-DRMM perspective, that follows the mathematical steps presented in Equation 4.11.

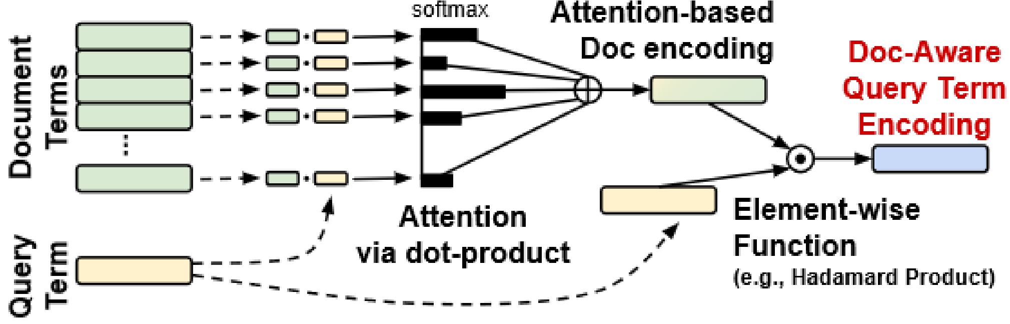


Figure 4.5: Architecture of the document query interactions follow the ABEL-DRMM. The image was withdrawn from the article [68].

Worth mention that this alternative will only replace the *matching histogram mapping* of the original DRMM.

- **POoled SImilariTy DRMM (POSIT-DRMM)** [68] this alternative states that the maximum and the average match between the query and document terms must be both considered by the model. However, the ABEL-DRMM formulation does not follow this assumption, since the attention summation hides the matching contribution of the terms matching, i.e, from the resulting encoding it is not possible to know if a single or multiple terms were matched with high similarity.

  To achieve the proposed idea the authors describe a max and k-max polling strategy over the query document term similarity. This way the maximum and the average match signal are kept, contrary to the ABEL-DRMM. Equation 4.12 shows the mathematical steps behind this idea.

$$\begin{aligned} a_{i,j} &= cos(c(q_i), c(d_j)) \\ a_i &= \underbrace{[a_{i,1}; ...; a_{i,j}]}_{concatenation} \\ \phi_P(q_i) &= \underbrace{[max(a_i); avg(k\text{-max}(a_i))]}_{concatenation} \end{aligned} \tag{4.12}$$

  Firstly the attention weight ($a_{i,j}$) for the i-th query term and j-th document term is computed by the cosine similarity (Equation 3.2). Then a 2-dimensional *Doc-Aware Query Term* vector ($\phi_P(q_i)$) is created using a pooling strategy, where the first dimension corresponds to the maximum value of the attention weight for the i-th query term and the second dimension is obtained by the average over the top k maximum values of the attention scores for the i-th query term.

As a final note, this alternative also uses the *context-aware* encoding of a term ($c(t_i)$) proposed in Equation 4.10 and it will also replace the *matching histogram mapping* in the original DRMM.

Both extensions were used in the previous year competition of the BioASQ challenge.

4.2.2 DeepRank

Following with the *Interaction Based* architecture, Pang *et al.* [69] proposed a new deep model for relevance ranking inspired by the human methodology. In their perspective, the model should emulate the human judgement process when looking for relevant documents, this process can be divided into three steps.

- **Detection** - Find passages on the document that are relevant to the query.
- **Measurement** - Measure the importance of each passage.
- **Aggregation** - Decide if the document is relevant based on the previous measures.

So the DeepRank model follows this process by implementing a **detection network**, followed by a **measurement network** and finally by an **aggregation network** that gives a relevance score.

Given a query and a document, the **Decision network** will find relevant passages to that query and for each query-passage will create a tensor, S, that captures their interactions. Based on *query-centric assumption* [70] the authors consider a passage to be relevant if it matches any token of the query, i.e, if a document token at position k matches any query token, the passage will be a sequence of tokens belonging to the interval $[k-7, k+7]$. Assuming that $\vec{x_i}$ and $\vec{y_i}$ are, respectively, the embeddings of the query and the document token at position i. So each entry in the tensor, S, correspond to the cosine similarity between these tokens, $S_{ij} = \frac{\vec{x_i}^T \cdot \vec{y_i}}{\|\vec{x_i}\| \times \|\vec{y_i}\|}$, $S \in]-1,1[^{(Q,P,1)}$. The authors also try to extend the S tensor by concatenating the query and document embedding, e.g, if the embedding dimension is represented by 200-vector and let Q and P be the respective length of the query and the passage, then the S tensor becomes with the following dimensions $S \in \mathbb{R}^{(Q,P,401)}$.

The **Measurement network** has the objective of transforming the tensor S into a scorable vector, i.e, the network will try to condense all the interactions that are present S into a fixed size vector $\vec{p}$. For that the authors proposed a CNN or a 2D-GRU, that is directly applied to each query-passage tensor. In the case of the CNN it is used a 3 by 3 kernel followed by a global max pooling layer. On the other hand, the 2D-GRU, proposed by the same authors in the following paper [71] is a special case of GRU that processes the tensor from top-left to bottom-right in a recursively way. Beyond that, the authors also concatenate to the final fixed size vector, $\vec{p}$, a function of the position of the passage relative to the document, $g(p)$, where g can be a constant, linear, reciprocal or exponential function.

Finally, the **Aggregation network** will first aggregate the vectors of the passages, $\vec{p}$, that share the same query token, resulting in the vector $t(\vec{w_i})$ where w_i corresponds to the token at position i in the query. Then the final score is computed using a term level aggregation over all the vectors $t(\vec{w_i})$. A GRU was chosen by the authors to aggregate the sequence of passages,

i.e, the GRU will transform the sequence of $\vec{p}$ aggregated by their shared token w_i into $t(\vec{w_i})$. The term level aggregation has one trainable weight per token and follows Equation 4.13.

$$s = \sum_{w_k \ k \in q} \underbrace{(\theta_k \times \vec{o})^T}_{1\times G} \cdot \underbrace{t(\vec{w_k})}_{G\times 1} \tag{4.13}$$

Here, θ_k is a trainable weight associated with the token k, $\underset{G\times 1}{\vec{o}}$ is a ones-vector with size G and the aggregation vector per query token, $t(\vec{w_k})$, as also the size of G.

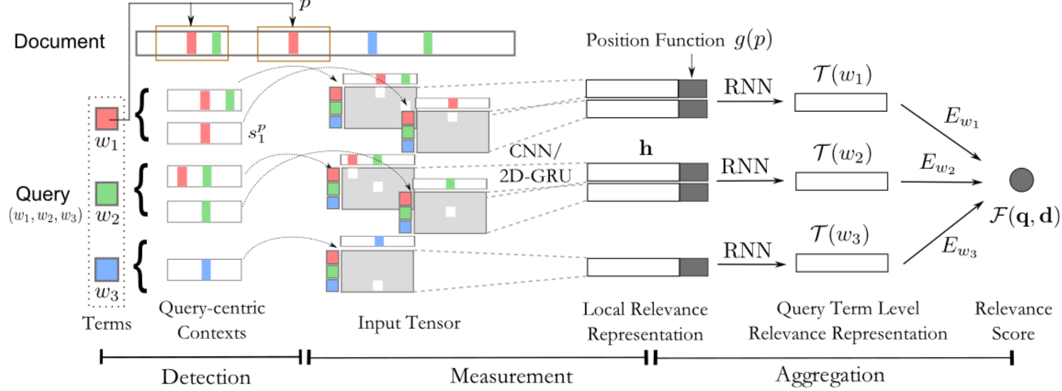


Figure 4.6: Complete visualization of the deeprank model with several steps of granularity. The image was withdrawn from the article [69].

Unlike the DRMM, the DeepRank model is a complete end-to-end neural network, since it is also capable of back-propagate the gradients to the embedding layer. It can be trained by any gradient descent algorithm and the authors decided to use the hinge loss 3.7, leaving open the possibility of using other types of losses. Figure 4.6 presents a visualization of the complete model were the different sub-networks are also identified. In their experiences, this model has shown an improvement over the state of the art DRMM and other strong baselines. Concrete comparison is shown at the end of this chapter.

4.2.3 A Hierarchical Attention Retrieval Model

At the time of writing the most recent neural ranking model in the literature is the HAR model proposed by Zhu *et al.* [72]. This model is *interacted based* and uses multiple attention mechanism to create a jointly abstract representation of the query and document, that is scored by a neural network.

Essentially, the HAR model uses the following two attention mechanism.

- **Self-Attention** - Creates a new representation of the input sequence by a weighted aggregation, already presented in 2.6.1.
- **Bidirectional Cross-Attention** - Captures the query-document interaction by computing the relevance of each query word with respect to each document word, and vice-versa.

Given two embedded input sequences, e.g query $\underset{m\times E}{Q} = \{\vec{q_1}, \vec{q_2}, ..., \vec{q_m}\}$ and document $\underset{n\times E}{D} = \{\vec{d_1}, \vec{d_2}, ..., \vec{d_n}\}$, the **Bidirectional Cross-Attention** mechanism, proposed in [73],

computes the attended relevance of each embedding in d_i with each embedding in q_j and vice-versa. Which correspond to the Document-to-Query (D2Q) attention and Query-to-Document (Q2D) attention, respectively. Another way to interpret is by considered that the attention occurs in two directions simultaneously, from document to the query as well as from the query to the document.

In terms of conceptualization, a similarity matrix $S \in \mathbb{R}^{n \times m}$ is firstly constructed, where each row corresponds to a document word embedding from D, each column corresponds to a query word embedding from Q, and each entry of the similarity matrix s_{xy} is described by Equation 4.14.

$$s_{xy} = \underset{1\times 3E}{w_c^T} \cdot \underbrace{[\underset{E\times 1}{d_x} ; \underset{E\times 1}{q_y} ; \underset{E\times 1}{d_x} \otimes \underset{E\times 1}{q_y}]}_{\text{concatenation}} \tag{4.14}$$

Here, w_c is a trainable vector that project the interaction vector to a single value and each interaction vector is given by the concatenation of the query and document embedding plus their element-wise multiplication, $\otimes$. Then the softmax operation, Equation 2.6, is applied over each row and column of the similarity matrix S, which gives, respectively, the D2Q and Q2D attention weights Equation 4.15.

$$\begin{aligned} S_{D2Q} &= softmax_{row}(S) \\ S_{Q2D} &= softmax_{col}(S) \end{aligned} \tag{4.15}$$

Intuitively, the query embeddings, Q, is weighted by the attention matrix S_{D2Q} resulting in the attended matrix A_{D2Q}, which tell us what are the query words that are most important to each document word. Likewise, the document embedding, D, is weighted by the attention matrix S_{Q2D} resulting in the attended matrix A_{Q2D}, which tell us what are the document words that are most important to each query word. Equation 4.16 shows how this computation is performed.

$$\begin{aligned} \underset{n\times E}{A_{D2Q}} &= \underset{n\times m}{S_{D2Q}} \cdot \underset{m\times E}{Q} \\ \underset{n\times E}{A_{Q2D}} &= \underset{n\times m}{S_{D2Q}} \cdot \underset{m\times n}{(S_{D2Q})^T} \cdot \underset{n\times E}{D} \end{aligned} \tag{4.16}$$

Finally, the attended vectors present on the attended matrices are multiplied by the document embedding D to complete the bidirectional flow, $\underset{n\times E}{D} \otimes \underset{n\times E}{A_{D2Q}}$ and $\underset{n\times E}{D} \otimes \underset{n\times E}{A_{Q2D}}$. Note that in this paper the cross-attention is performed on the document side, as shown in Figure 4.7.

In terms of network **architecture**, Figure 4.7, the model receive as input a query and a set of sentences that composes the document, both the inputs are represented as a sequence of tokens.

This model uses an embedding layer to transform each token into an embedding vector, then a bidirectional GRU is used as an encoder to add context-aware to the embeddings creating the context-aware vector u. This model uses two separated encoders, one for the

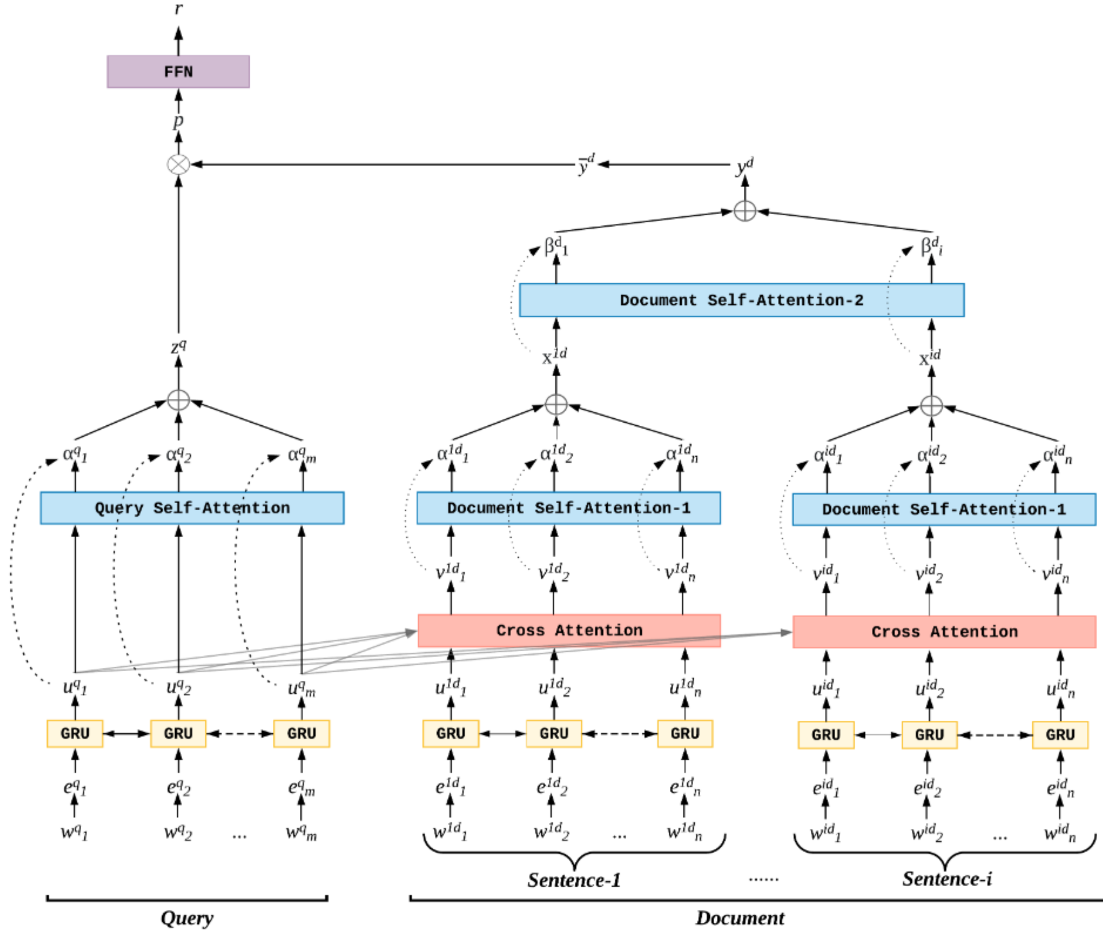


Figure 4.7: Complete visualization of the HAR. The image was withdrawn from the article [72].

query and other for the sentences, creating the respective context-aware vectors u^q and u^s. Focusing on the query side, a self-attention layer is then used to create a vector representation that condenses the most important information on the query, resulting in $z^q \in \mathbb{R}^{E \times 1}$. On the document side, cross attention is performed between context-aware vectors of the query and each sentence, creating the vector $v^{s_{i,j}}$ that corresponds to the cross attention output of the j-th token of the i-th sentence. After this, a two-level self-attention mechanism is performed to first create a vector representation of each sentence and then create a representation for the document corresponding to the vector $y^d \mathbb{R}^{4E \times 1}$. Finally, y^d is reduced to the same dimension of z^q by using a trainable projection vector and the jointly query-document representation is computed by multiplying z^q with y^{-d}, the resulting vector is fed to a three layer neural network that outputs the document score.

In an additional note, the authors added to the output of the cross attention the sentence context-aware U^s and the D2Q attend matrix A_{D2Q}. So the output matrix of the cross attention between the query and the i-th sentence corresponds to the matrix $\underset{n \times 4E}{V^{s_i}} = \underbrace{\{\underset{n \times E}{U^s}; \underset{n \times E}{A_{D2Q}}; \underset{n \times E}{U^s \otimes A_{D2Q}}; \underset{n \times E}{U^s \otimes A_{Q2D}}\}}_{\text{concatenation}}$ and $v^{s_{i,j}} \in \mathbb{R}^{4E \times 1}$.

In terms of optimization, they used the Adadelta [10] optimizer with a learning rate of

2 and the hinge loss function Equation 3.7. During training, for each query were selected six completely irrelevant documents and three partially negatives. The authors considered a partially negative document as a document that has some relevance to the query but does not contain the answer.

The authors also create a dataset for the healthcare domain (*HealthQA*), which they used to train and test the HAR model. They gather a total of 1235 articles and for each article, a total of six human annotators were encouraged to construct 1 to 3 simple questions that can be asked about that document. The final dataset contains 1235 articles and 7515 questions.

4.3 Summary

In this section is presented an *a priori* comparison of the previous models based on their definition and published results. Table 4.2 shown an overview of all the models. They are compared in terms of task, the needed of embedding, overall architecture, if is required training, loss objective and the inference time. Note that only the trainable models have a loss objective.

Table 4.2: Summary table that compares all of the presented models in terms of task, embedding, overall architecture, if is required training, loss objective and the inference time.

Models	Task	Pre-trainned embeddings	Architecture
AWE	Learn to Match	Any embedding representation	Representation based
DESM	Learn to Match	Embedding created with Word2Vec	Representation based
DSSM	Learn to Match	No, uses hash trick over BoW	Representation based
WMD	Learn to Match	Any embedding representation	not applied
DRMM	Learn to Rank	Any embedding representation	Interaction based
DeepRank	Learn to Rank	Any embedding representation	Interaction based
HAR	Learn to Rank	Any embedding representation	Interaction based
Models	**Require Training**	**Loss objective**	**Inference Time**
AWE	No	-	Fast (dot product)
DESM	No	-	Fast (dot product)
DSSM	Yes	Pointwise (cross-entropy)	Fast (dot product)
WMD	No	-	Slow (solve optimization)
DRMM	Yes	Pairwise (hinge)	Slow (run neural net over the collection)
DeepRank	Yes	Pairwise (hinge)	Slow (run neural net over the collection)
HAR	Yes	Pairwise (hinge)	Slow (run neural net over the collection)

Table 4.3 shows some comparatives results on different datasets withdrawn from the DeepRank [69] and HAR [72] articles. In the first case, the tests were performed on the MQ2007 [74] dataset[4]. On the second case, it was used the HelthQA dataset, as mention in the previous section, which is not publicly available.

On MQ2007 dataset the DeepRank get an upper hand when comparing to the other models in terms of MAP values. The same way, on the HealthQA dataset the HAR outperformed the other available models in terms of MRR values.

These results are only presented to give an intuition of what to expect from each model. However, the relative performance of these models should not be extrapolated when applied to biomedical data, since the domain and size of the data will be completely different.

[4]https://www.microsoft.com/en-us/research/project/letor-learning-rank-information-retrieval

Table 4.3: Summary table with some comparatives results of some neural models

Model	MAP on MQ2007	MRR on HelthQA
BM25 (baseline)	0.45	-
DSSM	0.409	-
CDSSM	0.364	0.645
DRMM	0.467	0.740
DeepRank	**0.497**	-
HAR	-	**0.879**

Additionally, in both cases, the implementation of the BM25, DSSM, C-DSSM and DRMM follow the default settings, which is also a bad indicator of the true potential of each model.

CHAPTER 5

Architecture and Implementation

This chapter firstly presents, with a high level of abstraction, the adopted architecture to tackle the biomedical information retrieval problem. Then, it is shown a more detailed view of each individual component and their implementation.

The retrieval system follows the architecture presented in Figure 5.1 and can be visualized as a pipeline. The inputs are the user queries and the collection of documents that will be searched. The outputs are the top ten most relevant documents or a set of relevant snippets from these documents.

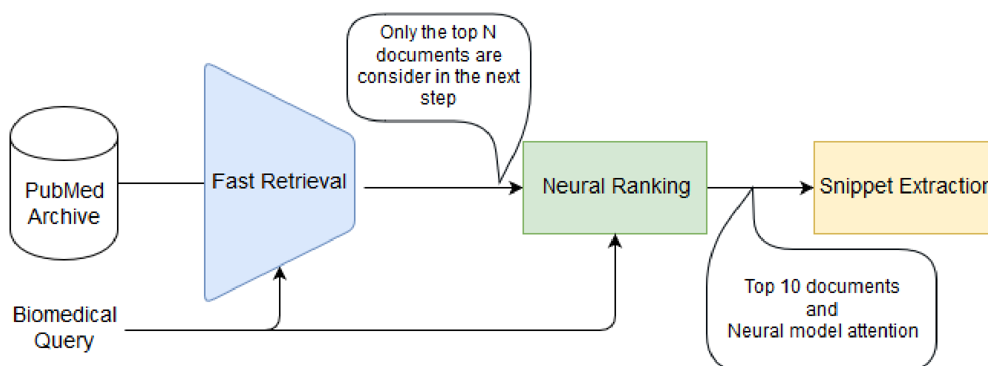


Figure 5.1: Overview of the principal modules of the proposed system.

This adopted pipeline can be reduced to three major tasks **filtering**, **ranking** and **extracting**. Which gives the name to each individual module, **Fast Retrieval**, **Neural Ranking**, and **Snippet Extraction**.

- **Fast Retrieval** - Select a set with N possible candidates to a query from the document collection.
- **Neural Ranking** - Uses a neural model to rank the possible N candidates and return the top ten with a higher score.

- **Snippet Extraction** - Highlights the most relevant snippets from each of the ten retrieved documents.

The idea behind this division is essentially due to the speculation that the neural model's inference times would be too higher to be applied to the full collection. So to cope with this issue, a module that will act as a filter was introduced before the use the neural model. The following section discusses the chosen technologies and after that, each module will be introduced and detailed.

5.1 Technologies and libraries

This dissertation follows a **hands-on** methodology. So all the neural and non-neural models were implemented from scratch in Python3.5.2[1].

5.1.1 TensorFlow

TensorFlow [75][2], developed by the *Google Brain Team*[3], is an open-source library[4] for distributed numerical and auto-differentiable computations, which is the basis for any implementation of deep learning algorithms. It was implemented in C++, yet a native Python API is available.

In *TensorFlow*, the mathematical computations are defined as a directed graph composed by a set of *nodes* and *edges*. Each *node* represents an operation or variable, while the edge, also called *tensor*, defines a data flow between the *nodes*. Additionally, it is possible to divide the computational graph into different chunks, which enable the parallelization across multiple Central Processing Unit (CPU), Graphics Processing Unit (GPU), and Tensor Processing Unit (TPU) or even the distribution across multiple devices.

On the left side of Figure 5.2, that is presented on *TensorFlow* article [75], it is shown an example of a direct graph created for the computation of the cost value C with respect to the operation $relu(W \cdot x + b)$. The operation of the cost value is omitted for sake of simplicity. On the right side, is shown an automatically generated graph for the computation of the gradients of C with respect to the set of inputs x. Furthermore, the gradients graph is created by backtracking the operation from C to x and for each operation, the partial derivatives along the backward passage are solved using the chain rule of differentiation.

This dissertation uses the version 1.9 of *TensorFlow*, which uses as default a *first define, then run* methodology. This implies that the computational graph must be firstly constructed and only then the computations can be performed. This methodology is no longer the default mode in the version 2.0-alpha of *TensorFlow*. Which now uses dynamic constructed graph's, enabling the graph definition while ruining.

[1]https://www.python.org/downloads/release/python-352/
[2]https://www.tensorflow.org
[3]https://ai.google/research
[4]https://github.com/tensorflow/tensorflow

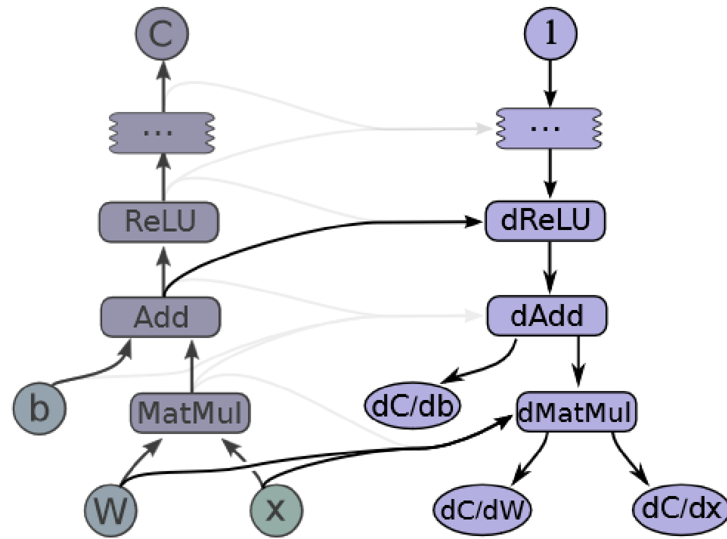


Figure 5.2: Generic example of a *TensorFlow* computation on the left and the auto-differentiable operation on the write. From [75]

Additionally, the *TensorFlow* package has sub-module[5] that implements the *Keras*[6] API, that offers a user-friendly interface for easily implement a great range of deep learning models.

Another popular candidate was the *PyTorch*[7] library by facebook. Essentially this library offers the same functionally as the *TensorFlow*. However, by default *PyTorch* uses dynamic constructed graphs opposed to *TensorFlow* version 1.9. The main reason to chose the *TensorFlow* was due to the greater community behind it.

5.1.2 Numpy

The Numpy[8] is an open-source[9] python library that offers an efficient way to operating with multidimensional numerical data (n-arrays).

5.2 Fast retrieval

The fast retrieval module was designed to act as a document filter capable of reducing the irrelevant documents while keeping the relevant documents to the query. For that, the following requirements were selected to guide the retrieval mechanism choice.

- **Efficiency** - The execution time during inference must be within the second scale.
- **Efficacy** - The N retrieved documents must contain relevant documents to the query while keeping N small.
- **Scalability** (less severe) - The execution time should not scale with the size of the collection.

[5] tensorflow.keras https://www.tensorflow.org/guide/keras
[6] https://keras.io/
[7] https://pytorch.org/
[8] https://www.numpy.org/
[9] https://github.com/numpy/numpy

The **Scalability** is considered a less severe requirement because it only ensures the system longevity on futures updates to the document collection. For that reason, this requirement should only be considered if the first two requirements are met.

Based on the previous set of requirements, a mathematical score, $s(d)$, can be defined as shown in Equation 5.1, which offers a systematic methodology for choosing the ideal retrieval mechanism over a pool of candidates.

$$s(d) = \begin{cases} recall@N(d) & \triangle t <= T \\ 0 & \triangle t > T \end{cases} \tag{5.1}$$

Here, $\triangle t$ is the elapsed inference time in seconds, T is a maximum acceptable time in seconds and d correspond to the set of top N documents retrieved. The score is based on the value of $recall@N$ for the set of retrieved documents if their elapsed time is at most T. The intuition behind this formulation is that the **number of relevant documents returned by the retrieved mechanism must be maximized**. However, it is also important to **minimize the total number of returned documents** N, since the inference times of the neural models is directly proportional to N. This optimization problem may not be trivially solved[10], so to alleviate this, multiples runs with different values for N were taken into consideration and based on that, a suitable value for N was chosen. Then, after fixing the value for N, the retrieval mechanism with the highest score, $s(d)$, will be chosen to power this module.

From all the retrieval mechanism presented in Chapters 3 and 4, the eligible models for testing were, **BM25**, **AWE**, **AWE with TF-IDF** and the **DSSM** model. These models were chosen based on their potential to fulfill the previous requirements. In the following subsections will be discussed their individual implementation.

5.2.1 Implementation of BM25

As previously mentioned, this dissertation follows a hands-on methodology. As such, this subsection addresses the implemented technical details of the ranking function with the BM25 weight scheme. However, it is worth mentioning that popular search engines like Elastic Search[11] could be used instead.

Recalling Section 3.1.3, the ranking function, Equation 3.5, computes a matching score given a query-document pair. A naive implementation would be directly applying the equation to every pair, which is an inefficient solution. Therefore, an enhanced alternative would be to construct an inverted index, where each vocabulary term records a list of documents that contain it, recall Figure 3.1.

The diagram presented in Figure 5.3 shows the adopted flow to create an inverted index over a large collection of documents. Initially, all the documents in the collection are divided into sets, that have approximately the same size. Each of these sets is indexed in parallel,

[10]This affirmation is based on the fact that the values of the $recall@N$ and N are directly proportional, so the maximization of the *recall* and minimization of N will end up in a contradiction

[11]https://www.elastic.co/

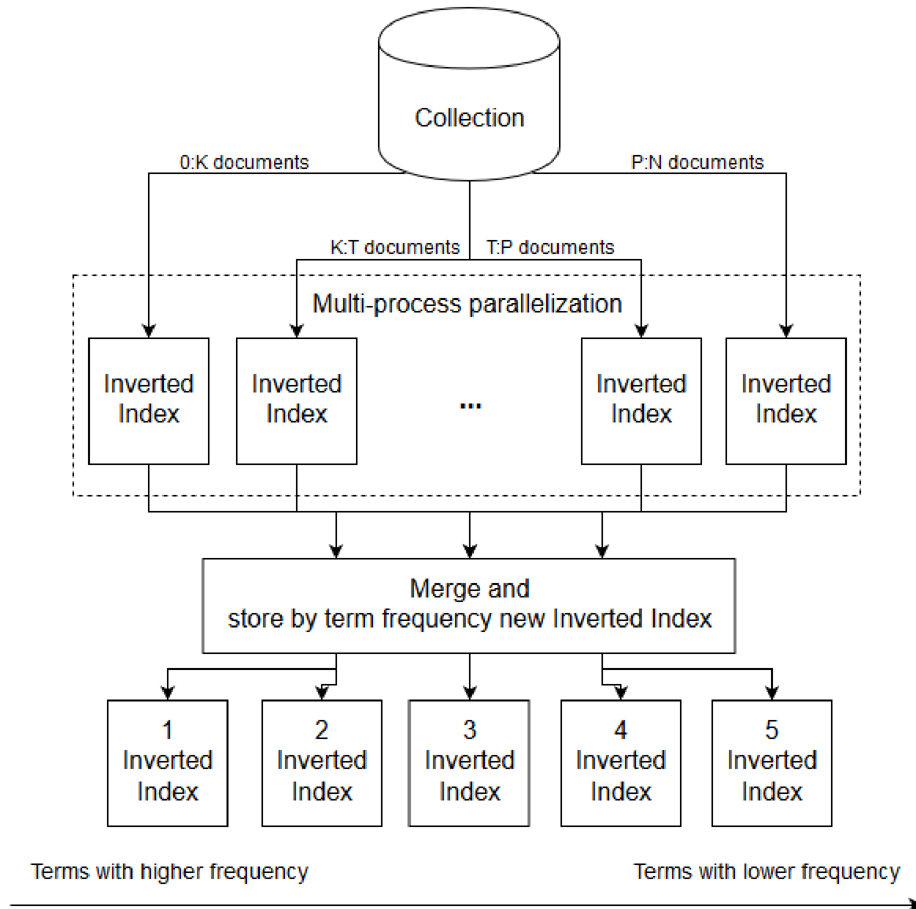


Figure 5.3: Flow diagram for the creation of the inverted index

which originates an inverted index per set. The second step, consists of organizing the inverted index by frequency term, i.e, the most frequent terms are grouped into the same inverted index, where the total number of documents in each group is limited by a predefined threshold. This ensures that all inverted indexes have approximately the same size in disk. For example, the inverted index with the most frequent terms (*1 Inverted Index*) contains fewer terms than the next inverted index (*2 Inverted Index*), but both have approximately the same size in disk.

The motivation for creating multiple inverted indexes is related to the high memory footprint needed for the complete inverted index. This implies that for a given query, different inverted indexes will be load from disk into memory, which increases the execution time. However, as each inverted index is organized by term frequency, in theory and on average, the number of inverted index loads per query is minimized, therefore the execution time is reduced.

In general, the **training process** for this retrieval mechanism, left side of Figure 5.4, consists in building the inverted index described in the previous paragraphs and by precomputing the BM25 weights for each term. For the last one to occur, the BM25 Equation 3.5 needs to

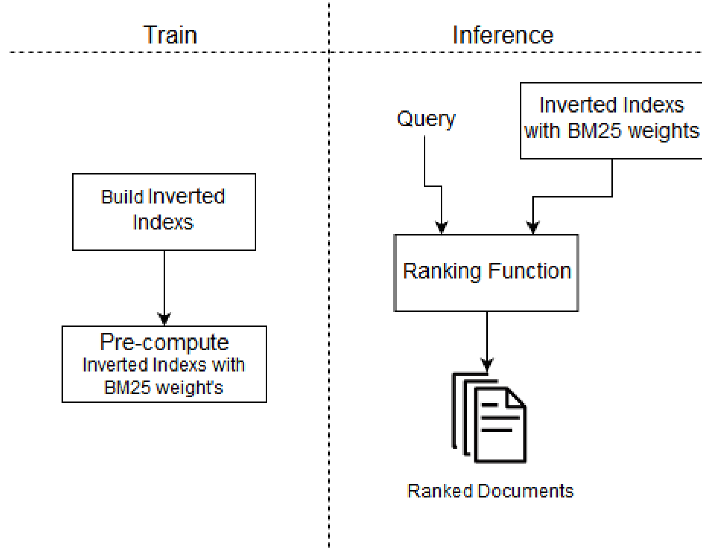


Figure 5.4: On the left is presented the Flow diagram for the training process and on the right is presented the Flow diagram for the inference process

be simplified, in order to not be dependent on the query term, resulting in Equation 5.2.

$$weight(t,d) = \underbrace{idf_t}_{\text{first part}} \times \underbrace{\frac{(k_1+1)tf_{t,d}}{k_1(1-b+b(|D|/avg_{|D|}))+tf_{t,d}}}_{\text{second part}} \tag{5.2}$$

Here, the third part of the equation was unconsidered, which is not severe since the queries are usually small. This way for each term in the inverted index the associated BM25 weight can be precomputed.

The **inference process**, on the right in Figure 5.4, creates a ranked list of documents, given a query and the inverted indexes following Equation 5.3.

$$score(q,d) = \sum_{t_q\ \in\ Q} weight(t,d) \tag{5.3}$$

The precomputation of the BM25 weighting scheme for each term ensures that the order of complexity of calculating $weight(t,d)$ is $O(1)$ and consequently the complexity of $score(t,d)$ is $O(|Q|)$, where $|Q|$ represent the number of query terms. So the complexity of this retrieval model must be inferior[12] to $O(|Q| \times |D^+|)$, where $|D^+|$ is the total number of documents that contains terms from the query Q, i.e, $|D^+|$ represents the total number of documents returned by the retrieval model. This complexity analysis ensures that the *scalability* requirement is respected, since $O(|Q| \times |D^+|)$ does not directly depend on the number of documents in the collection.

In Table 5.1 is shown a list of the chosen hyperparameters, where the k_1 and b values were chosen according to the literature [40], [41].

[12]Since $O(|Q| \times |D^+|)$ assumes that each query term appear in all the returned documents, which is false in practice.

Table 5.1: Hyperparameters chosen for the BM25 retrieval mechanism

Hyperparameters	
Number parallel processes	20
Number of documents per Inverted Index	309 146 814
k_1	1.2
b	0.75

From now and until the end of this dissertation the term *BM25 model* will be used to referring this retrieval model.

5.2.2 Implementation of AWE and AWE with TF-IDF

The **training process** implemented by the AWE and AWE with TF-IDF weighting is described in Algorithm 1. Essentially, consists of constructing a document embedding matrix, D_e. Where the i-th row corresponds to the i-th document embedding obtain by applying the AWE or AWE with TF-IDF, Equations 4.1, 4.2, to the document tokens.

Algorithm 1: Train algorithm for the AWE and AWE with TF-IDF retrieval mechanism.

Input : Document collection (D)
Output : Normalized document embedding matrix (D_e)

1 **for** $i \leftarrow 0$ **to** $|D|$ **do**
2 $\quad D_e[i] \leftarrow \text{awe(D[i])}$; `Equation 4.1 or 4.2`
3 **end**

Similarly, the **inference process** is described by Algorithm 2, which for a given set of queries (Q) computes their AWE representation. Then the cosine similarity between the queries representation and the representation of the documents is calculated using the vectorized implementation shown in Equation 5.4. Finally, the top N documents with the highest score are returned by the algorithm.

$$cos(\underset{|E|\times|Q|}{Q}, \underset{|D|\times|E|}{D}) = \frac{\underset{|D|\times|E|}{D} \cdot \underset{|E|\times|Q|}{Q}}{\underset{|D|\times 1}{\|C\|} \otimes \underset{1\times|Q|}{\|Q\|}} \tag{5.4}$$

This formulation computes the score for all the documents in the collection, D, given a set of queries, Q, where $|Q|$ correspond to the number of queries. The size of the embedding vector is represented by $|E|$.

Algorithm 2: Inference algorithm for the AWE and AWE with TF-IDF retrieval mechanism.

Input : Normalized document embedding matrix (D_e), Set of queries (Q)
Output : Top N documents with the highest score

1 **for** $i \leftarrow 0$ **to** $|q|$ **do**
2 $\quad Q_e[i] \leftarrow \text{awe(Q[i])}$; `Equation 4.1 or 4.2`
3 **end**
4 $rank \leftarrow \cos(Q_e, D_e)$; `Equation 5.4`
5 top($rank$,N)

The inference algorithm has a complexity order of $O(\overbrace{|Q| \times |avgQ|}^{\text{awe complexity}} + \overbrace{|D| \times |E| \times |Q|}^{\text{dot product complexity}})$. Here, the first term corresponds to the complexity of computing the AWE for the set of queries, where $|Q|$ represents the total number of queries and $|avgQ|$ correspond to the average length of the queries. The second term shows the order of complexity of the dot product operation

that is used by the cosine function, where $|E|$ corresponds to the size of the embedding dimension. However, when the algorithm only uses a single query the previous complexity order can be simplified to $O(|avgQ| + |D| \times |E|)$ since $|Q| = 1$. This complexity analysis shows that the *scalability* requirement is not respected, because it directly scales with the size of the collection. However, as will be shown in the next Chapter, in practice the dot product complexity is smoothed by the efficient mathematical libraries.

5.2.3 Implementation of DSSM

Recalling Section 4.1.3, the DSSM is a neural network that learns how to project the queries and documents into the same low-dimensional space. The implemented network architecture follows Figure 4.2 with the exception of the *word hashing* layer, that was not implemented, because it requires to have an in memory matrix with the dimensions, $|V| \times |V_t|$, size of vocabulary ($|V|$) by the size of the tri-gram vocabulary (V_t). Given this matrix size, it is impractical to keep it in memory, so an *on-the-fly* solution was implemented instead, i.e, the tri-gram representation of a sentence is computed before fed to the network.

The neural network tensor diagram is presented in Figure 5.5, wherein a) shows the DSSM model architecture and in b) shows the training architecture on top of the DSSM model. The blue boxes represent graph operations, more specifically in this case *TensorFlow* operations, while the arrows are the tensors connecting the different operations.

Following the implemented model architecture in **a)**, the DSSM network takes as input a tri-gram vector representation of a sentence (document or query). Then, three fully connected layers, each with *hyperbolic tangent* activation, will project the tri-gram vector into a 128-dimensional space.

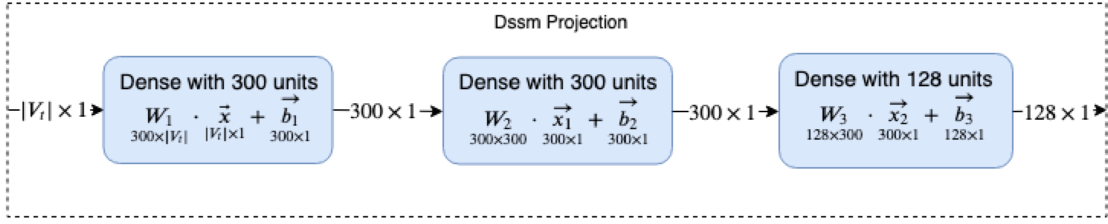
Recalling that the idea behind this model is to learn how to project query-document pairs that are relevant to a similar 128-dimensional vector. The training that is shown in **b)**, consists in compute the relevance probability for all the query-document pairs. However, this is impractical to do for every pair in the collection, so an approximation is made by using one relevant document and n irrelevant documents per query (NEG). Then, given the approximated distribution and a *true distribution*, the *cross-entropy* loss is calculated and the SGD optimizer will minimize such loss. Following the diagram in **b)** the *true distribution* corresponds to the fixed vector $\{1, 0, 0, 0, 0\}$ if $n = 4$.

The **training process** implemented here is similar to Algorithm 1, with the additional step of training the DSSM model and use them to make the document projection on line 2. As a result, the matrix $D_e \in \mathbb{R}^{(|D| \times 128)}$ is created and each row corresponds to the precomputed representation of a document by the DSSM model.

Likewise, the **inference process** is also similar to Algorithm 2, with the respective change on line 2, in order to use the DSSM model to get the projection of the query.

The complexity order of the inference process, $O(\overbrace{O(dssm) \times |Q|}^{\text{dssm complexity}} + \overbrace{|D| \times |E| \times |Q|}^{\text{dot product complexity}})$, is also similar to the AWE since the base algorithm is the same. The first term corresponds to the complexity of computing the projection for all the queries and the second term corresponds to the cosine similarity complexity between the queries and all the documents. So, the *scalability*

a) dssm model



b) dssm training architeture

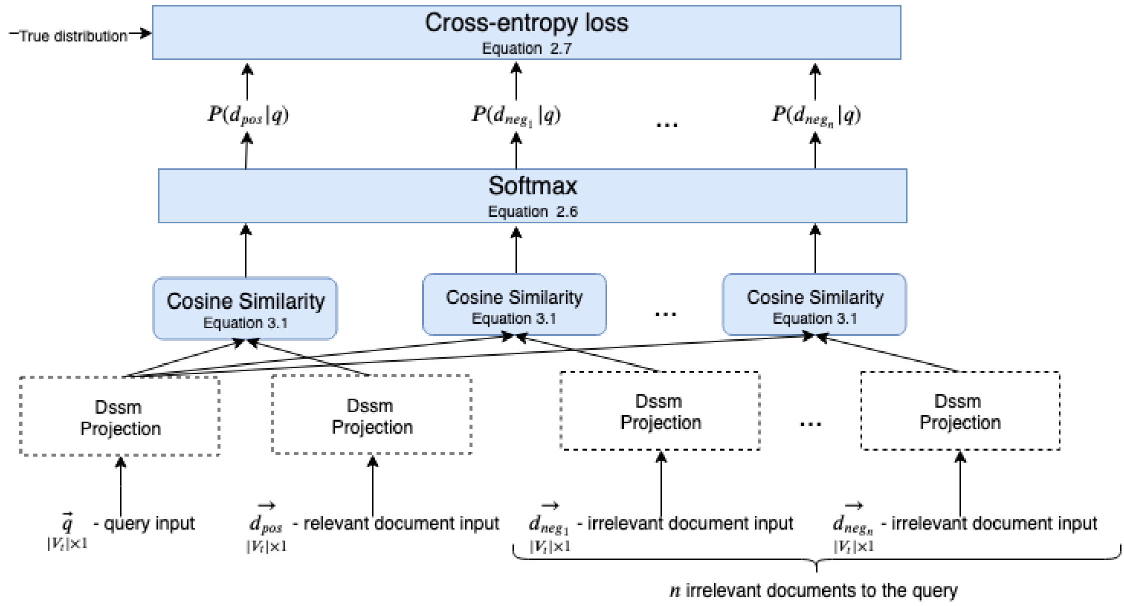


Figure 5.5: Tensor diagram of the dssm model in a). In b) tensor diagram created to train the dssm model. The blue color represents graph computations that were supported by the *Keras* API, the arrows are tensors.

requirement is also not respected because the complexity order directly scales with the size of the collection $|D|$. Based on the comparison of the complexity is also possible to infer that this method will be theoretically the slowest.

Table 5.2: Hyperparameters chosen for the DSSM

Hyperparameters	
$\lvert V_t \rvert$	48482
Activation function	tanh
Optimizer	SGD
learning rate	0.01

All the chosen hyperparameters are based on the original model and are presented in the Table 5.2. Note that, for a tri-gram vocabulary of $|V_t| = 48482$, the number of trainable weights of the DSSM becomes $48482 \times 300 + 300 + 300 \times 300 + 300 + 300 \times 128 + 128 = 14673728 \approx 14.6M$ trainable weights. So, in theory, this model will be prone to easily overfit, due to the high

number of parameters.

5.3 Neural ranking

The neural ranking module was designed to rank the top N documents, in a way that relevant documents are ranked higher than non-relevant ones. This module implements a neural retrieval mechanism that will return the top ten most relevant documents to a query. To accomplish this the following requirements must be taken into consideration.

- **Efficacy** - The 10 retrieved documents must contain relevant documents to the query.
- **Snippet based** - The neural architecture of the model must be designed around the idea that *"a document is a composition of snippets"*. This way it is possible to directly extract the relevant snippets of a relevant document, as will be explained more ahead.

The reason why **efficiency** was not taken into consideration is due to the fact that the number of documents to rank (N), already ensures that these models will have an acceptable inference time. Similarly, the **scalability** requirement was also dropped, since it can not be applied because the number of documents (N) is already fixed.

The decision of including a **Snippet based** requirement reduces the available pool of possible neural models. Nevertheless, that decision is motivated by the premise that *a good retrieval result should be the outcome of a set of relevant snippets present on a document.* A second motivation to consider this requirement is that allows the extraction of theses relevant snippets in the next module.

$$score(d) = map@10(d) \tag{5.5}$$

Similar to the fast retrieval model, Equation 5.5 is used to systematically test and find the best neural retrieval mechanism to be implemented by this module. In this case, the score is given by the map@10 metric and d are the N documents previously retrieved.

The eligible neural retrieval models, presented in Chapter 4, were **DeepRank** and **HAR** since are the only ones that respect the **Snippet based** requirement. In the following subsections are presented their implementation with the help of tensor diagrams, that for sake of simplicity the **batch/data dimension was not included**. Additionally, the **training** and **inference** process directly corresponds to the train and inference done by the deep model.

5.3.1 Implementation of DeepRank

The implemented DeepRank model, Section 4.2.2, follows the author's architecture, summarized in Figure 4.6. Firstly, let us define Q as the maximum number of query tokens, P as the maximum number of passages per query token, S as the maximum number of snippet tokens, and E as the size of the embedding vectors. So the model receives as input **a query**, **a set of snippets** respectively aggregated by their query term and absolute **position of the snippet** concerning the document. The snippets are aggregated at the input level to simplify the aggregation step, which will appear later on. Then the model is divided into three

complementary networks, the detection network, the measure network, and the aggregation network, that operate sequentially over the input.

Given a query and a set of snippets per query term, the objective of the **Detection network** is to create an interaction matrix between them, where each entry is computed by the dot product of their embedding vectors. Since both embeddings are normalized the dot product corresponds to the cosine similarity between the query and the snippet tokens.

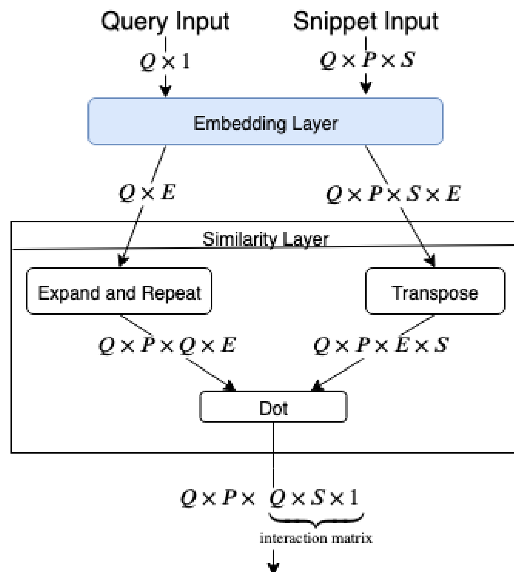


Figure 5.6: DeepRank detection network. Q is the maximum number of query tokens, P is the maximum number of passages per query token, S is the maximum number of snippet tokens, and E is the size of the embedding vectors. The blue boxes correspond to native *Keras* layers. Correspondent *TensorFlow* graph can be visualized in Figure B.1.

In Figure 5.6 is presented the tensor diagram of the detection network, that is mapped to *TensorFlow*. In terms of network, first the *embedding layer* maps each token to the respective embedding vector and them the *similarity layer* creates the interaction matrix. For the *embedding layer* the *Keras Embedding Layer*[13] was used, while the *similarity layer* was implemented by extending *Keras Base Layer*[14], the code can be visualized in Appendix C Code 1. It is worth to mention that most of the queries or snippets are padded to the correspondent maximum length, which implies that most of the interaction matrices are full of zeros. This *zero matrices* must be ignored during the network, for that masking layers are utilized to ignore them.

The **measure network** captures the most relevant matching signals for each interaction matrix, then all the resulting signals of the same query term are aggregated into a final representation.

The tensor diagram, presented in Figure 5.7, describes the operations of the computational graph implemented in *TensorFlow* for the measure network. First, a *2d convolution* with 3×3 kernel is applied to the interaction matrix resulting in F *feature maps*, where F define the

[13]https://www.tensorflow.org/versions/r1.9/api_docs/python/tf/keras/layers/Embedding
[14]https://www.tensorflow.org/versions/r1.9/api_docs/python/tf/keras/layers/Layer

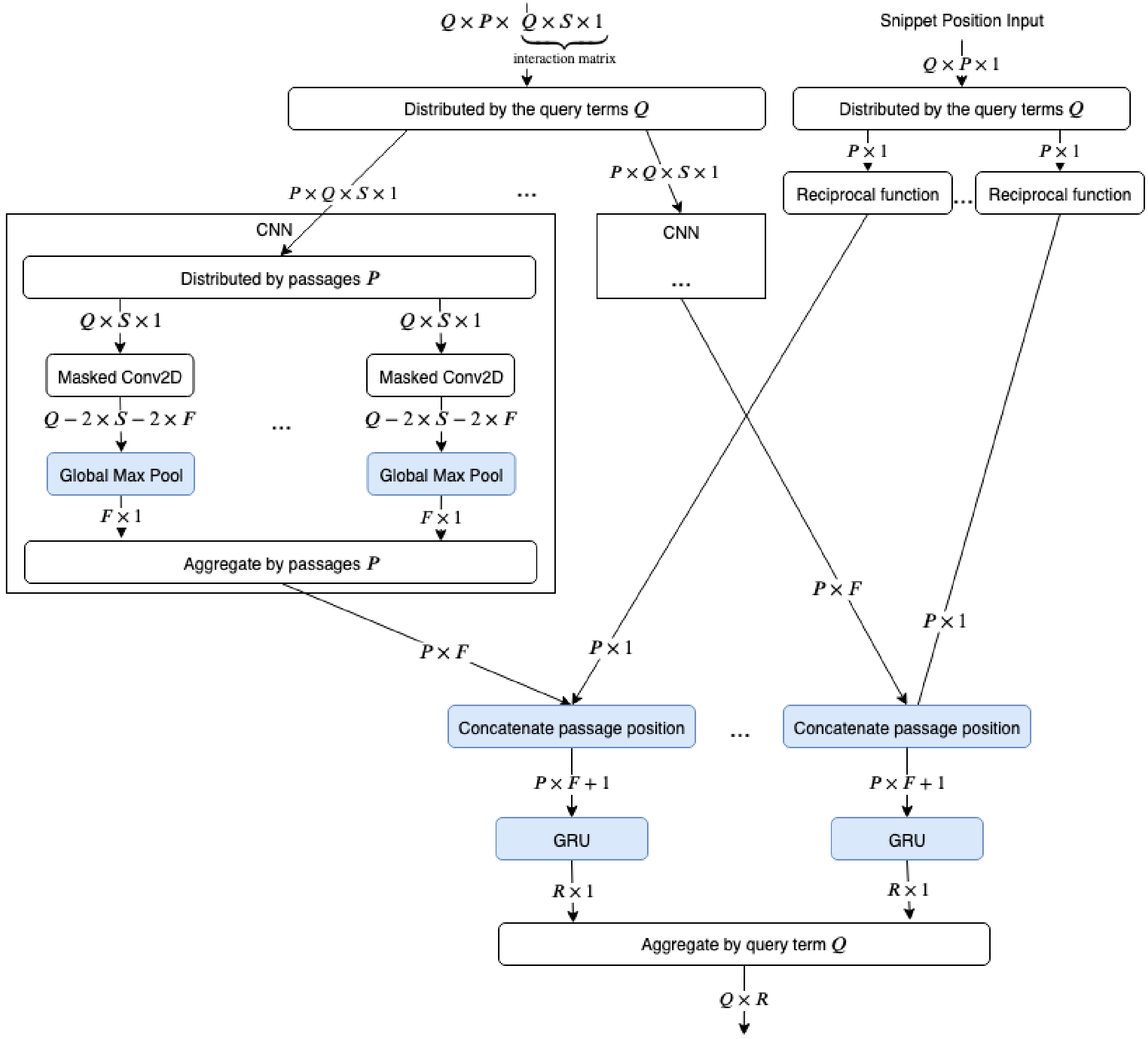


Figure 5.7: DeepRank tensor diagram of the measure network. F number of filters and R output dimension of the GRU. The blue boxes correspond to native *Keras* layers. Correspondent *TensorFlow* graph can be visualized on Figure B.2

number of filters used. Then a *Global Max Pool*[15] layer will select the higher value of each feature map, resulting in a vector representation, $\vec{p}$, with dimension F for each interaction matrix.

Without loss of generality, note that the layer *Masked Conv2D* correspond to a custom layer, code in Appendix C Code 2, because the masking mechanism in *Keras* was too generic to be used in this model. So, for every interaction matrix with *full zeros* the *Masked Conv2D* will ignore them by multiplying the output by zero. Also note that when a convolution is applied to zero matrices, the result is the *bias* value for each entry in the resulting feature map.

Continuing with the diagram flow, each absolute position of the snippet is transformed by the reciprocal function, $g(x)$, Equation 5.6, and is concatenated to the vector $\vec{p}$. Then a

[15] https://www.tensorflow.org/versions/r1.9/api_docs/python/tf/keras/layers/GlobalMaxPool2D?hl=en

GRU will aggregate the vectors $\vec{p}$ that belong to the same query term, w_i, creating the final vector representation $t(\vec{w_i})$. This last step was included in the measure network, rather than in the aggregation network as proposed in the original model since it facilitates the network assembling and does not interfere with the model performance.

$$g(x) = \frac{1}{x+2} \tag{5.6}$$

Finally, the **aggregation network** will group the vectors $t(\vec{w_i})$ by their respective query term, w_i, producing $\vec{t}$. This step must take into consideration the importance of each query term (w_i), i.e, the vector $t(\vec{w_i})$ is weighted by the importance of the word w_i. Then the final vector $\vec{t}$ is used to produce a real number that corresponds to the query-document score.

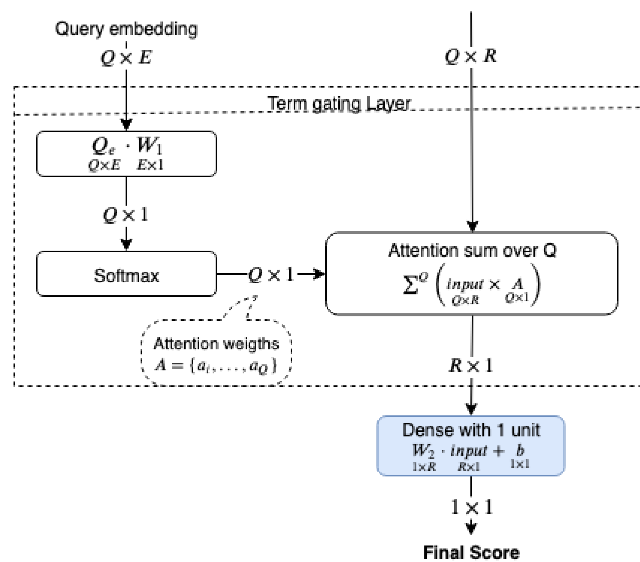


Figure 5.8: DeepRank tensor diagram of the aggregation network. The blue boxes correspond to native *Keras* layers. Correspondent *TensorFlow* graph can be visualized on Figure B.4

In Figure 5.8 is presented the tensor diagram corresponding to the implemented computational graph for the aggregation network. The *Term gating* layer is a custom layer, Appendix C Code 3, that will infer the importance of each query term (attention weights) based on their embedding vector. Then, it computes a weighted sum of $t(\vec{w_i})$, that is fed to a *Dense*[16] layer that outputs a final score.

Note that, the gating mechanism implemented here differs from the original mechanism described in Equation 4.13 since the authors used a trainable variable per vocabulary token. However, the biomedical data has a rich vocabulary, which will result in an extremely large trainable vector. So the implemented gating mechanism uses the embedding of each token to directly compute the gating weights, this way as shown in the diagram, it only requires a trainable vector with the size of the embedding vector.

The sequential connection of the three sub-networks previously presented, constitute the DeepRank model, as shown in the tensor diagram in Figure 5.9.

[16]https://www.tensorflow.org/versions/r1.9/api_docs/python/tf/keras/layers/Dense

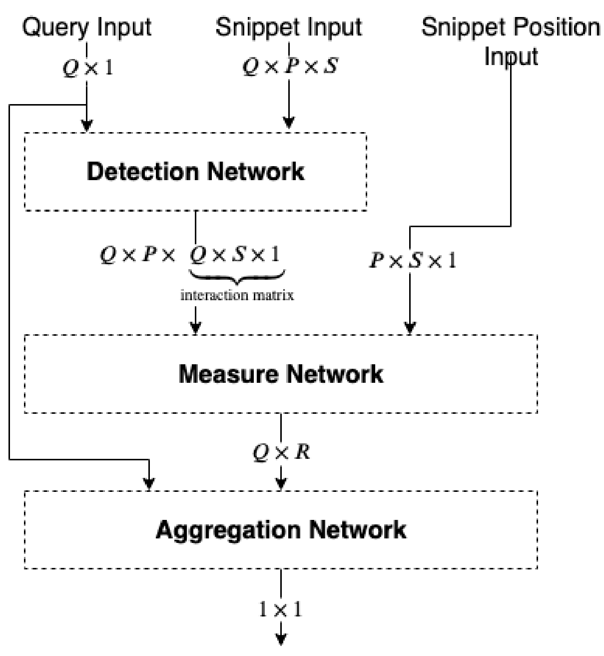


Figure 5.9: DeepRank tensor diagram.

To train this model is used the *hinge loss*, Equation 3.7, as suggested by the authors. This loss compares the score of a relevant query-document pair directly to the score of an irrelevant pair, so no implicit labels are required, i.e, each training sample correspond to the following tuple *(query-positive document, query-negative document)*. As a consequence, it was necessary to implement this loss as a tensor operation and manually add it to the graph, since it was not supported by the *tensorflow.keras* library. This process can be visualized in the tensor diagram Figure 5.10, where the two DeepRank models correspond to the same model, i.e, they share the same training variables and operations.

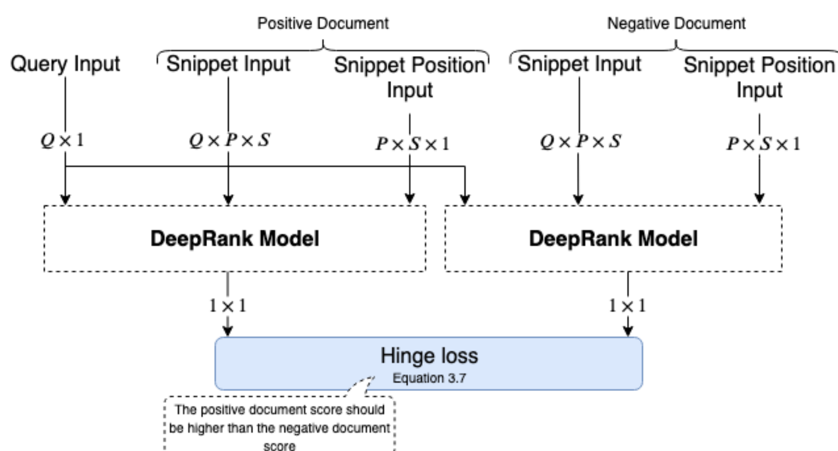


Figure 5.10: DeepRank tensor diagram of the training architecture.

Recalling the system pipeline, this model occurs after the **fast retrieval method** with the purpose to re-rank its results, which means that the available documents for the negative

sampling must be withdrawn from these results and not from the entire collection since it is more similar to what the model should expect during inference time. However, these results already carry strong matching signals, ensured by the**fast retrieval method**, which can hamper the model training. So, as suggested in [76], some completely irrelevant documents were also added as part of the negatives documents. As a result, a negative document can be sampled from:

- A set with *partially relevant negative* documents, that corresponds to the documents returned by the **fast retrieval method** that are not relevant to the query.
- A set with *irrelevant negative* documents, that corresponds to the documents from the collection that do not share any matching signal with the query.

Table 5.3 shows all the chosen hyperparameters concerning the implementation and training of the model. The resulting model with this configuration of hyperparameters has a total of 29.1 thousand trainable weights. Note that the weights of the embedding matrix are discarded since they are not trained. The reason to not train the embedding matrix is to keep the focus on the model, so that can be compared with others.

Table 5.3: Hyperparameters chosen for the DeepRank

Hyperparameters	
Q - number max of query tokens	13
P - number max snippets per query token	5
S - number max of snippet tokens	15
E - Size of embedding vector	200
F - number of convolutional filters	100
R - number of recurrent units	56
Number of *Partially relevant negative* samples	2
Number of *Irrelevant negative* samples	3
Activation function	Selu [77]
Optimizer	AdaDelta [10]
Learning rate	2
Regularization	$l_2 = 0.0001$
Train embedding	No

DeepRank variation

Although the DeepRank respects the **snippet based** requirement, it is challenging to extract the weights of each snippet with respect to the document final score, i.e, it is challenging to know what are the snippets that most contribute to the final score, because the result of the GRU aggregation is a vector, which only the model can interpret. Inspired by the inner-workings of the attention networks [78], a solution to this problem is to replace the GRU snippet aggregation by a self-attention snippet level aggregation, were the attention weights will be directly related to the relevance of each snippet. The tensor diagram presented in Figure 5.11 shows the replacement of the GRU layer by the self-attention layer in the **measure network**. From now on, this variation will be refereed as **Self-Attn-DeepRank**.

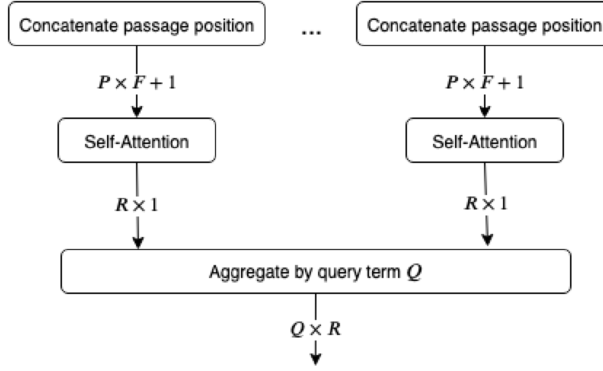


Figure 5.11: Snippet of tensor diagram of measure network with self-attention layer.

The implemented self-attention layer follows Equation 2.15 and the *align model* is described by Equation 5.7. The code of this layer is shown in Appendix C Code 4 and the respective tensor diagram in Figure 5.12.

$$a_i = \underset{1\times R}{v^T} \cdot tanh\left(\underset{R\times F+1}{W} \cdot \underset{F+1\times 1}{\vec{p_{(i)}}}\right) \tag{5.7}$$

Here, the attention weight a_i correspond to the weight of the i-th passage vector ($\vec{p_{(i)}}$) for each query term.

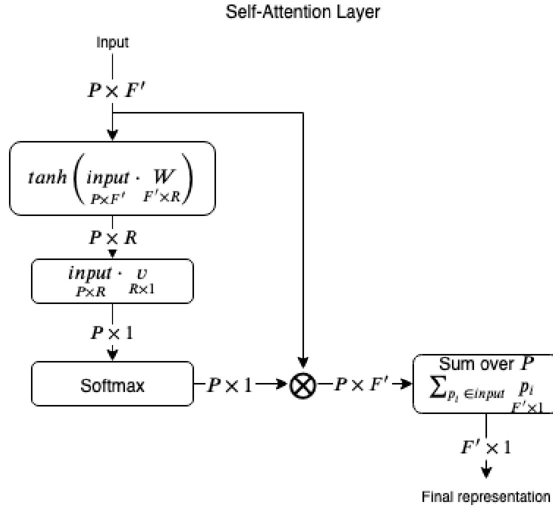


Figure 5.12: Tensor diagram of self attention layer. Correspondent *TensorFlow* graph can be visualized in Figure B.5

In this tensor diagram, for sake of simplicity, it is used the dimension F' that corresponds to $F+1$, i.e, $F' = F+1$. R is an intermediate dimension and represents the size of the attention vectors. The flow of this diagram directly follows Equations 2.15 and 5.7.

Finally, this variation was trained with the same architecture and hyperparameters of the original. The *Self-Attn-DeepRank* model has a total of 11.6 thousand trainable weights, which

compared with the original is less than half of the training weights. So, this reduction as the potential to improve the model's inference times.

5.3.2 Implementation of Hierarchical Attention Retrieval Model

The implemented HAR model follows the author's description and Figure 5.13 presents the respective tensor diagram that is implemented in *TensorFlow*. The blue boxes correspond to native *Keras* layers that were used, while the other boxes correspond to custom operation or layers that were implemented from scratch.

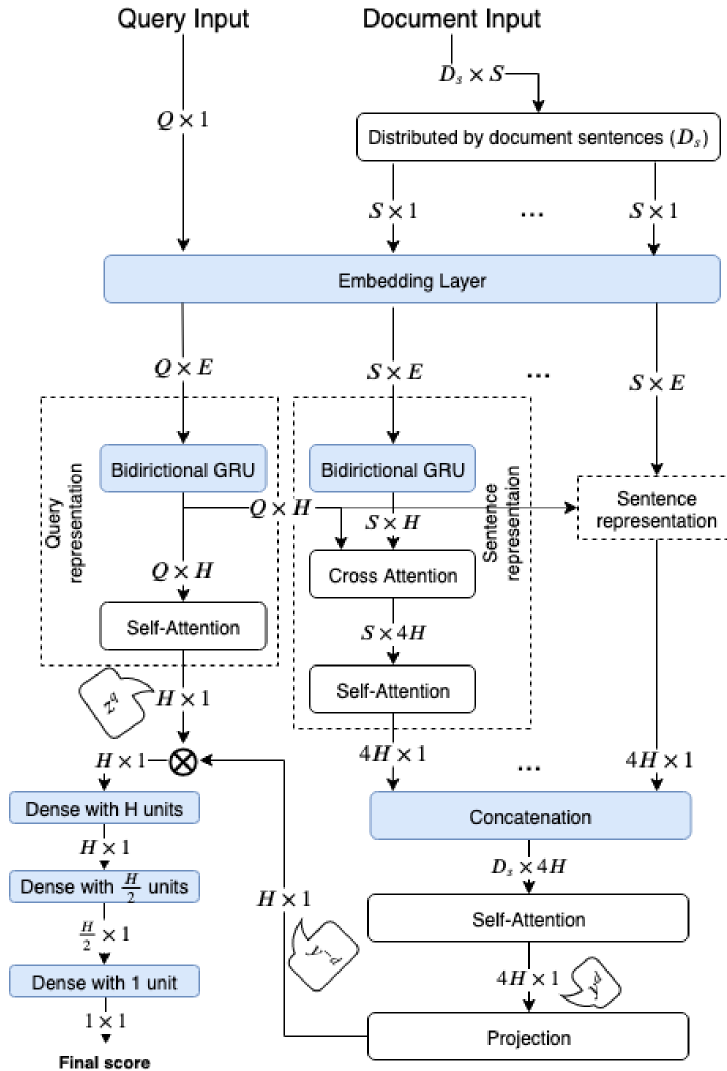


Figure 5.13: Tensor diagram of the complete HAR model. The blue boxes correspond to native *Keras* layers. Correspondent *TensorFlow* graph can be visualized in Figure B.6.

To better understand the tensor diagram of the model, lets us define Q as the maximum number of query tokens; D_s as the maximum number of sentences in a document; S as the number maximum of tokens in a sentence; E as the size of the embedding vector and H as an inner dimension of the model that define the number of GRU units and attention size. Additionally, can be useful to recall the data flow described in Section 4.2.3 to get a better

interpretation of this tensor diagram.

Note that the *Bidirectional GRU* that is used to encode the query embeddings is different from the one used to encode the sentence embeddings. Additionally, the implementation of the *Self-Attention* Layer was already presented by the tensor diagram in Figure 5.12, while the implementation of the *Cross Attention* Layer is described in the tensor diagram, Figure 5.14.

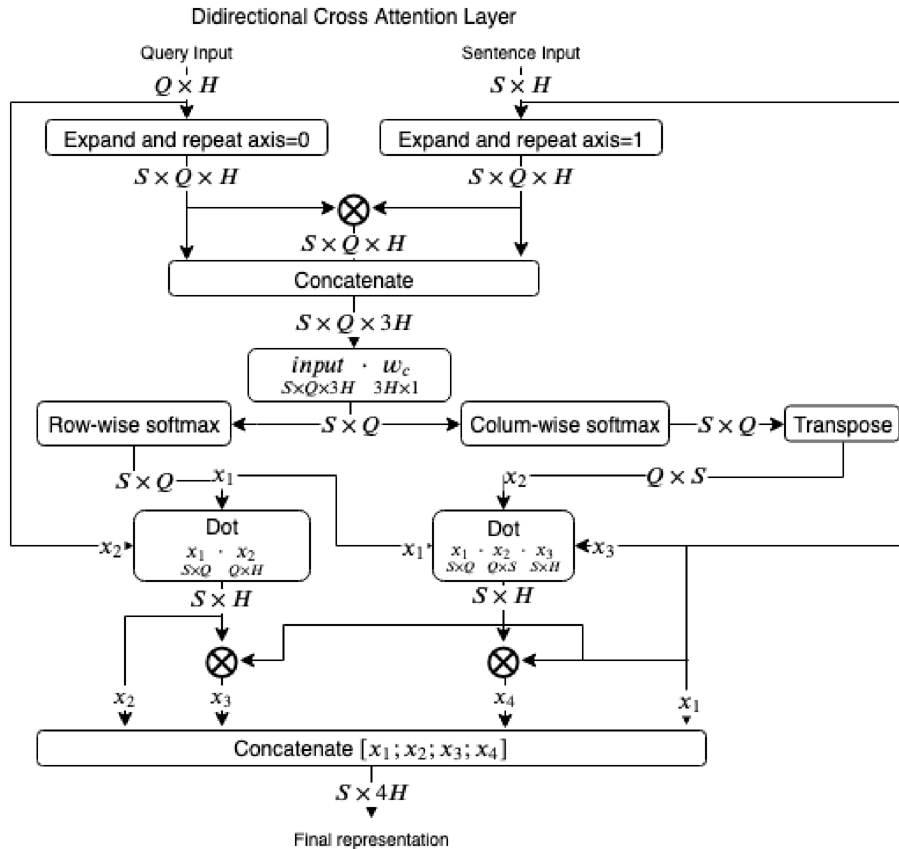


Figure 5.14: Tensor diagram of the Cross Attention Layer. Correspondent *TensorFlow* graph can be visualized in Figure B.7.

This diagram shows the operations needed to create the similarity matrix, that is then used to compute the D2Q and Q2D attention matrices, which corresponds to Equations 4.14, 4.15 and 4.16.

The same exact train architecture used by the *DeepRank* model was utilized to train the HAR model, which correspond to the implementation of the hinge loss as a tensor operation, shown in Figure 5.15.

Table 5.4 shows the adopted hyperparameters to train this model and following this configuration the resulting model has a total of 221.2 thousand trainable weights. So, this model has almost eight times more trainable weights than the *DeepRank*, which will become more easily to overfit. To cope with this problem, it was added dropout layers between the dense layers and l_2 regularization to the loss.

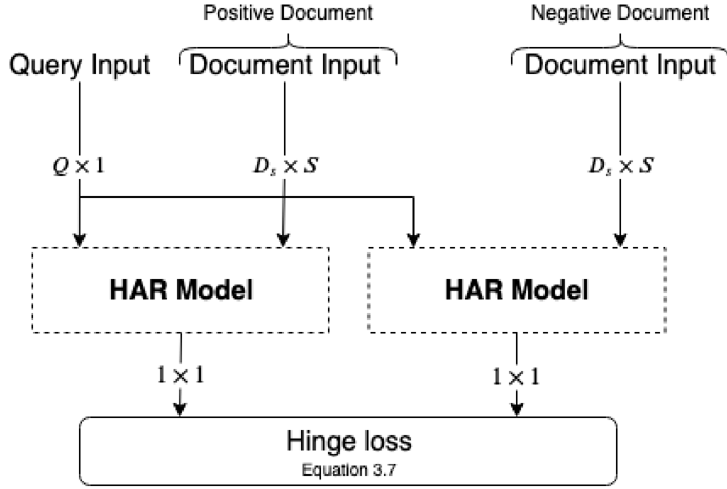


Figure 5.15: Tensor diagram of HAR of the training architecture

Table 5.4: Hyperparameters chosen for the HAR model

Hyperparameters	
Q - Number max of query tokens	13
D_s - Number max of document sentences	40
S - Number max of sentence tokens	13
E - Size of embedding vector	200
H - Attention and Bi-GRU dimension	100
Number of Partially relevant negative samples	3
Number of Irrelevant negative samples	6
Activation function	Selu [77]
Optimizer	AdaDelta [10]
Learning rate	2
Regularization	$l_2 = 0.001$ and dropout ($dr = 0.2$)
Train embedding	No

5.4 Snippet extraction and visualization

The snippet extraction module was designed with the intent to discriminate the information inside a document since, usually, only small portions of a document are relevant to a query. So the idea is to **highlight** the most important information inside a document to a given query. This module takes advantage of the **snippet based** requirement in the previous module, to facilitate the extraction of the most relevant information in the form of snippets.

This snippet extraction was only implemented to the *Self-Attn-DeepRank* and *HAR* models since the original version of the *DeepRank* uses a GRU layer to aggregate the snippets to a single vector, which requires a more deep analysis to understand how the most relevant snippets correlate with the produced recurrent vector.

For the ***Self-Attn-DeepRank*** model, this correlation can be directly extracted by looking at the attention weights produced during the snippet aggregation. Besides that, the attention weights of each query term are also taking into consideration, since the snippets are aggregated

by these query terms. So, given a document and a query let us define the attention weights of the self-attention layer as $\underset{Q\times P}{A_{qs}^{dr}}$ and the attention weights of the term-gating layer as $\underset{Q\times 1}{\vec{a_q}^{dr}}$, where Q is the number of query terms and P the number of snippets per query term. Note that, the self-attention layer is applied to each query term, so A_{qs}^{dr} corresponds to the concatenation of the snippet attention weights for all the query terms, i.e, A_{qs}^{dr} represents a concatenation of the local attention of the snippets for each query term $\underset{Q\times P}{A_{qs}^{dr}} = \{\underset{P\times 1}{\vec{a_{qs_0}}^{dr}}; ...; \underset{P\times 1}{\vec{a_{qs_Q}}^{dr}}\}$. So, to get a global attention weight with respect to each snippet an element-wise multiplication is performed between A_{qs}^{dr} and $\vec{a_q}^{dr}$, Equation 5.8.

$$\underset{Q\times P}{A_s^{dr}} = \underset{Q\times P}{A_{qs}^{dr}} \otimes \underset{Q\times 1}{\vec{a_q}^{dr}} \tag{5.8}$$

The element-wise multiplication, $\otimes$, can be seen as a normalization over each row of A_{qs}^{dr} with respect to the attention given for each query terms. Now each weight in A_s^{dr} will **approximate** the global importance that is given to each snippet. In practice, only the top ten most important snippets are extracted from A_s^{dr}, so a more normalization step is required to ensure that the attention weight is in probabilistic distribution format, Equation 5.9.

$$\underset{10\times 1}{\vec{a_s}} = \frac{\overbrace{top(A_s, 10)}^{10\times 1}}{\underbrace{\sum_{x \,\in\, top(A_s,10)} x}_{1\times 1}} \tag{5.9}$$

Here, the matrix A_s^{dr} can be seen as a flat vector and the *top* as a function that will retrieve the ten higher values. Now, the vector $\vec{a_s}$ corresponds to the attention weights of the ten most relevant snippets in the perspective of the model.

In the case of the ***HAR*** model, the snippet importance can be directly inferred from the attention weights of the document level self-attention layer. So, given a document and a query let us define the attention weights of the document level self-attention layer as $\underset{D_s\times 1}{a_d^{\vec{har}}}$ and the attention weights of the query self-attention as $\underset{Q\times 1}{a_q^{\vec{har}}}$, where Ds is the maximum number of snippets. As previously mentioned, only the top ten most important documents are extracted, which means that the previously normalization step, Equation 5.9, is also applied to $a_d^{\vec{har}}$ resulting in $\underset{10\times 1}{a_d^{\vec{har}}}$.

The snippets are displayed with a blue background color in a *hsl*[17] format, where the *lightness* (l) value is inversely proportional to the value of the attention weight, i.e, higher attention weights have lower light values (darker color). The idea is to facilitate the distinction of the relative importance of each snippet, this way the most important snippets are presented by a progressively darker blue. In a similar way, the query attention weights are presented by a red background color in a *hsl* format.

[17]https://www.w3schools.com/colors/colors_hsl.asp

5.5 System as web application

This section presents web application that exposes the system pipeline in an online environment that can be accessed by a web browser. This is an extension of the current developed work that was not originally planned. Figure 5.16 shows an overview of the implemented architecture in order to expose the system pipeline. The front-end was built using React[18], because offers a way to encapsulate the code in components. So in practice, the search mechanism was encapsulated by a React Component[19] that can be reused in different web pages. For the back-end was used the Flask[20] micro-framework that works in Python, which facilitates the integration with the created system since it is also written in Python.

In a more detailed way, the Flask Server offers a route to access the *html* pages and another route (*/api*) that executes the system pipeline for a given query. In the */api* route the query is sent in JSON format on the body of a POST request.

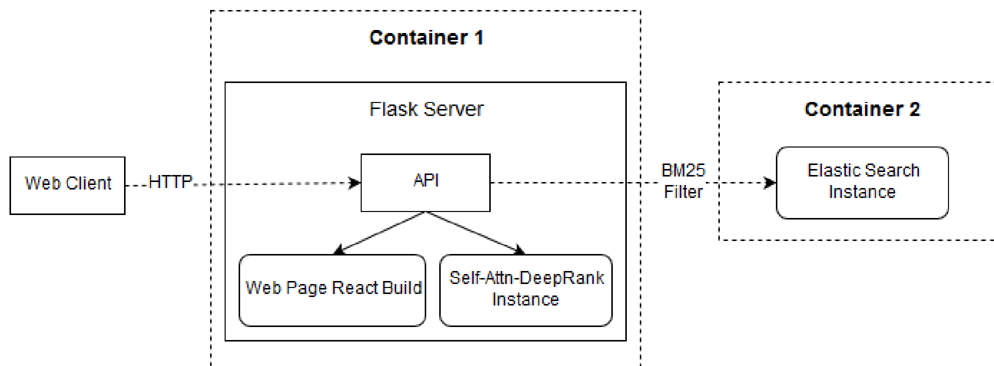


Figure 5.16: Overview of the web application components.

As an additional note, the BM25 version that runs on elastic search was used since has better inference times as will be shown later on.

[18]https://reactjs.org/
[19]https://reactjs.org/docs/react-component.html
[20]https://github.com/pallets/flask

CHAPTER 6

Experiments and Results

This chapter will present the experiments and the results for each of the three modules and the overall experimental result for the system.

6.1 Biomedical data

As mentioned before, the proposed system will be applied to the biomedical data, however, it is generic enough to be adapted to other domains. The major challenging of working with biomedical data is the extensive and specialized vocabulary that the models must take into consideration, i.e, comparing with more generic retrieval tasks, beyond the ordinary English vocabulary it is added a large specialized vocabulary, which can range from molecules names to DNA sequences. This large vocabulary presents an even greater challenge to the neural models since it increases the difficulty of creating distributed vector representations of the words (embeddings).

6.1.1 BioASQ dataset

BioASQ[1] is a public competition, where the organizers promote annual challenges on biomedical semantic indexing and question answering (QA), currently, this competition is sponsored by Google. The biomedical semantic index, also known as task *A*, consists of annotating articles with classes from MeSH[2], which is not addressed in this dissertation. On the other hand the question answering, task *B*, is subdivided into two phases; phase *A* involves the information retrieval problem, where is asked to retrieve the most relevant documents, snippets, RDF triples or concepts to a given question; phase *B* involves the extraction or generation of an answer for a query given the previous retrieved information. For both tasks, the BioASQ organizers make available a training set that the competitors can use to train their models. More specific for the document retrieval task *B* phase *A*, that is the focus of this dissertation, the training data consist on a set of queries with the correspondent relevant articles selected by

[1] http://bioasq.org/
[2] https://www.ncbi.nlm.nih.gov/mesh

biomedical experts from an annual PubMed/MEDLINE repository. In terms of competition, the BioASQ provides a set of test queries, in pre-established date, and the results can be submitted, online, in their platform to be evaluated.

This year is the seven edition of the BioASQ challenge. The released training set contains a total of 2747 queries and the article collection is the 2018 PubMed/MEDLINE repository. Beyond that, the organizers released five test sets[3] with 100 queries each. For each query, the ten most relevant articles should be retrieved from the collection. In terms of evaluation, the retrieved articles are first evaluated using the following metrics *Average Precision*, *Recall*, *F1*, *Map@10* and *GMAP*. In a later phase, the biomedical experts will inspect each non-relevant article to see if it should be considered as relevant.

As an additional note, since the third edition of the BioASQ challenge the used *Map@10* metric, Equation 6.1, was slightly modified according to BioASQ Evaluation Guidelines [79].

$$BioASQ_{MAP@10} = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \underbrace{\frac{1}{10} \sum_{k=1}^{10} Precision@k \times rel(k)}_{\text{Average Precision}} \tag{6.1}$$

Here, the MAP is computed at ten documents, since is the maximum number of documents allowed by the BioASQ. Another important detail is the fraction $\frac{1}{10}$ that is also fixed, this results in an additional penalization when compared to the original MAP function, because the number of relevant documents in the dataset is on average less than ten for each query.

6.1.2 PubMed

PubMed[4] is a free online search engine, available since 1996, that currently contains about 30 million citations of biomedical literature from MEDLINE[5], life science journals, and online books. Being the MEDLINE database the primary source of indexed citations. Annually the MEDLINE/PubMed[6] repository is released in *xml* format with all the citations indexed until that respective year. The publicity available EDirect[7] software was used to download the 2018 annual repository, that is the document collection for the BioASQ dataset. The 2018 MEDLINE/PubMed repository has almost 31 millions articles of which only 19 million are composed by title and abstract.

Table 6.1: Number of articles in 2018 MEDLINE/PubMed dump.

MEDLINE/PubMed	
Total number of articles	30 862 349
Number of articles with title and abstract	18 824 355

[3]http://participants-area.bioasq.org/Tasks/7b/phaseB/
[4]https://www.ncbi.nlm.nih.gov/pubmed
[5]https://www.nlm.nih.gov/bsd/medline.html
[6]https://www.nlm.nih.gov/databases/download/pubmed_medline.html
[7]https://dataguide.nlm.nih.gov/edirect/overview.html

In this section will be addressed the preprocessing steps of the queries and articles, which involves the tokenization and the embedding vectors.

6.2.1 Tokenization

The tokenization is a process that transforms a sentence into a sequence of tokens, where each token should correspond to a word, a number or a specific term/symbol. Normally, a tokenizer uses a parser with a set of pre-established rules to transform a sentence into the tokens. For example, a simple rule can be: "split by spaces" and when applied to the sentence *"Bob is smart"* will result in the following sequence of tokens *"Bob"*, *"is"*, *"smart"*.

Since the tokenization process occurs before the creation or training of the retrieval models, recalling Figure 3.1, the quality of these tokens will directly influence their performance. So the correct choice of tokenizer is of great importance, however, it is challenging given the extensive biomedical vocabulary. The following works [80], [81] also reinforces this idea and try to alleviate the burden of this choice by showing several standard tokenizers applied to biomedical data. An example, it is the chemical substance *"4-epoxy-3-methyl-1-butyl-diphosphate"* that can result in different tokens depending on the tokenizer rules.

This dissertation uses two tokenizers, the **Bllip**[8], that will be designated as tokenizer 1 and the **Regex**, that will be the tokenizer 2.

The **Bllip** [82] is a two stage statistical parser, publicly available[9], that produces a tokenized tree with Part-of-Speech (POS) tags. However, for this dissertation, only the sentence splitting rules are used to make the tokenization. In the end, this tokenizer will lower-case the sentence and then apply the splitting rules, that tries to keep all the words intact, only some characters like commas, parenthesis and punctuation are ignored or transformed. With this parser the tokenization of *"4-epoxy-3-methyl-1-butyl-diphosphate"* will result in a unique token (*"4-epoxy-3-methyl-1-butyl-diphosphate"*). Additionally, each produced token is *stemmed* using *nltk stem*[10]. The stemming is a process for reducing a group of similar words to the same root using heuristics, e.g, *walked*, *walks*, *walking* are reduced to the word *walk* by removing the suffixes. The stemming will sacrifice the loss of the words syntax in order to reduce the vocabulary size.

The **Regex** tokenizer utilizes an alphanumeric regular expression to split the sentence, this is a more simplistic approach and is intended to create small vocabularies without modifying the tokens. For example, with this tokenizer the complex word *"4-epoxy-3-methyl-1-butyl-diphosphate"* will result in the sequence of tokens *"4"*, *"epoxy"*, *"3"*, *"methyl"*, *"1"*, *"butyl"* and *"diphosphate"*.

Both tokenizers were applied to almost 19 million articles that composes the collection and Table 6.2 shows the number of unique tokens produced by each tokenizer, i.e, the vocabulary size. In a first analysis, the number of unique tokens is extremely large, which supports the

[8] http://bllip.cs.brown.edu/resources.shtml#software
[9] https://github.com/BLLIP/bllip-parser
[10] https://www.nltk.org/api/nltk.stem.html

Table 6.2: Size of the resulting vocabulary for the bllip and regex tokenizer

Tokenizer	Vocabulary size
Bllip tokenizer 1	11 108 738
Regex tokenzer 2	4 291 793

idea that biomedical data has an extensive vocabulary. To better understand these numbers a second analysis was made, where it was added a minimum word frequency restriction to the produced vocabulary. In other words, the size of the vocabulary was analysed as a function of the minimum token frequency, which mean that only the tokens that appear at least a specific number of times were included in the tokenization. In Figure 6.1 it is shown the *bllip* vocabulary, on the left, and the *regex* vocabulary, on the right, both as a function of a minimum token frequency from 1 to 20.

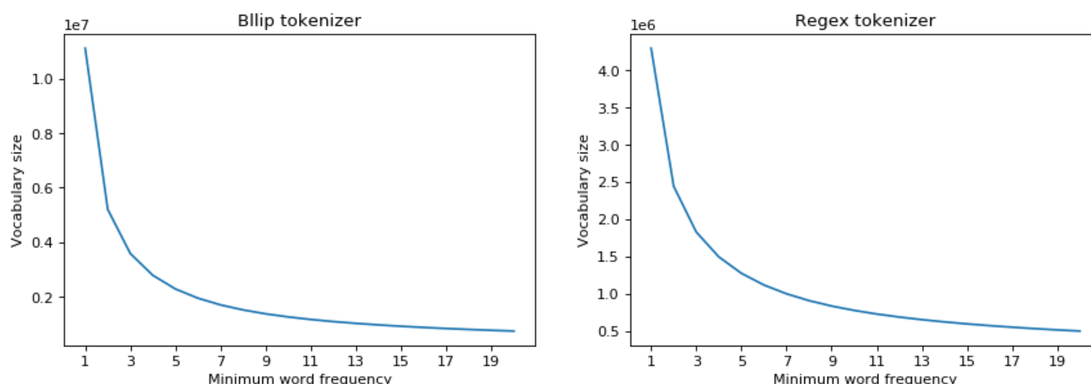


Figure 6.1: Comparison of the minimum number of occurrences of word for each tokenizer.

In both cases, the function follows a reciprocal behaviour, which indicates that most of the tokens are extremely rare in the collection. For example, in the case of the *bllip* tokenizer with a minimum token frequency restriction of 20, the resulting vocabulary has only 744 thousand unique tokens, as shown in Table 6.3.

Table 6.3: Size of the vocabulary with minimum token frequency restriction for the bllip and regex tokenizer

Tokenizer	Vocabulary size with restriction at 20	Size reduction (%)
Bllip tokenizer 1	744 567	1490
Regex tokenizer 2	502 263	850

A common practice in Natural Language Processing (NLP) tasks is to only use the most frequent tokens since it is computational more efficient. For example, instead of using the *bllip* vocabulary, it is preferable to use the *bllip* vocabulary with a restriction at 20, because it produces a smaller vocabulary. However as mentioned before, for the retrieval task the most important signal is the exact match between the query and the document tokens, which is directly injured by the reduction of vocabulary. Additionally, the matching signal between less frequent terms is usually the most important one to be captured since is associated with

more important words. For this reason, the complete vocabulary without restriction must be chosen to process the data for the retrieved models.

6.2.2 Collection and BioASQ statistics

The two tokenizers were applied to the entire collection and in Figure 6.2 is shown the distribution of the number of tokens in the documents (document length), using a histogram with the bin set to 500.

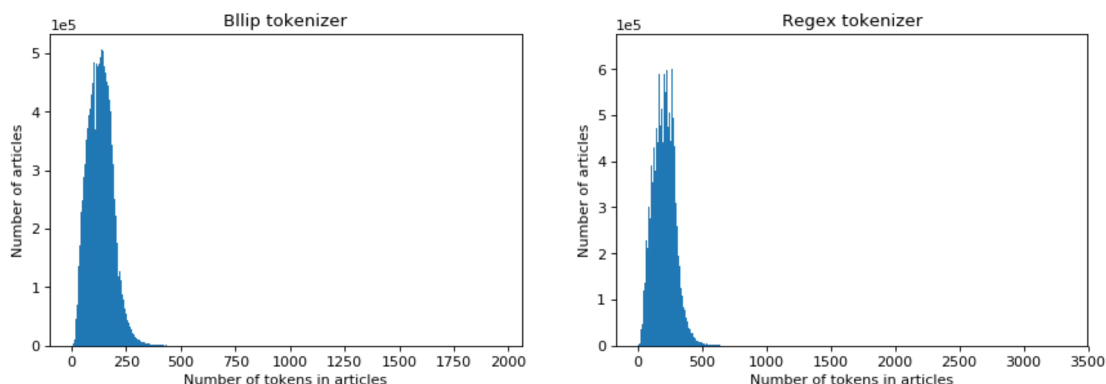


Figure 6.2: Distribution of the number of tokens by documents for both tokenizers. Histogram constructed with $bin = 500$

Both distributions resembles a gausian distribution and when compared, the *bllip* on the left with *regex* on the right, it is clear that the *bllip* produce documents with lesser tokens. This is explained by the fact that the *regex* tokenizer usually splits complex words in multiple tokens, recalling the *"4-epoxy-3-methyl-1-butyl-diphosphate"* example. On average, the documents tokenized with *bllip* have 133 tokens. On the other hand, the documents tokenized with the *regex* have 208 tokens.

The same analysis was made for the BioASQ queries on the training data and the same conclusion is verified, as shown in Figure 6.3. On average the queries tokenized with *bllip* have 5 tokens and the queries tokenized with *regex* have 9 tokens.

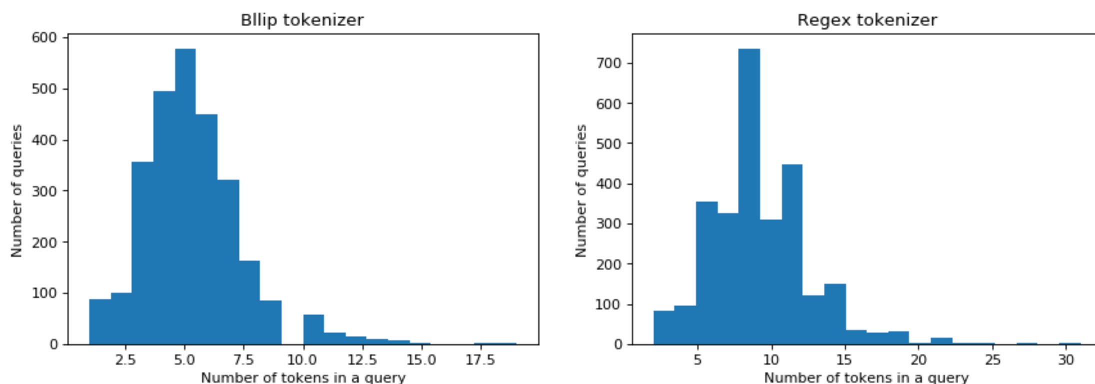


Figure 6.3: Distribution of the number of tokens by queries for both tokenizers. Histogram constructed with $bin = 20$.

6.2.3 Embeddings

For both tokenizer, the respective embedding vector was obtained by using the pre-trained *fasttext* model published by Zhang *et al.* [83]. For each token in the vocabulary, the *fasttext* model was able to compute the correspondent embedding vector, as shown in Section 3.2.5. The authors trained the *fasttext* model on biomedical data to create a 200-dimension embedding vector for each word and sub-word. For the training data, they used the PubMed/MEDLINE collection and clinical notes from MIMIC-III Clinical Database [11]. The model is available on github[12].

Alternatively, the *word2vec* algorithm could also be used to learn the embedding vectors. However, due to time limitation, this alternative would not be feasible. Furthermore, would even be less robust, since any change in the vocabulary would imply the rerun of the *word2vec* algorithm.

6.3 Evaluation methodology

Before the evaluation of the general system, an evaluation of each individual module is performed, in order to decide the respective retrieval model that will power each module. The chosen dataset was from the BioASQ competition.

In a first stage, during development, an 80% train and 20% validation split methodology for the BioASQ training data was followed resulting in 2498 and 549 queries respectively. The validation split was used to find the best fast retrieval models and neural models. In a second stage, the BioASQ testing data from this year was used to evaluate the overall system and compare with other submission/systems.

To facilitate the testing and validation between different models, a testing framework was created. This will ensure that all the models are validated/tested in a similar way. In practice, this framework is accomplished by ensuring that all the created models implement a common interface, *ModelAPI*, that will offer a systematic way of testing and a unified way of interacting with all the neural an non neural models. The *ModelAPI* interface, presented as a snippet in the Appendix C Code 6, covers all the basic tasks that each model should need, i.e, doing predictions, training if applicable, save/load their state and perform an evaluation following a set of predetermined metrics.

6.3.1 Hardware

All the training and tests were run on a server, Table 6.4 shows the server specifications.

[11] https://physionet.org/physiobank/database/mimic3cdb/
[12] https://github.com/ncbi-nlp/BioSentVec

Table 6.4: Server specifications

Server		
CPU	Intel(R) Xeon(R) CPU E5-2670 v3	
	cores/threads	12/24
	base/turbo clock	2.3/3.1 GHz
RAM	192 GB	

6.4 Fast retrieval Results

Following the methodology present in Section 5.2 the first step was to choose a suitable value of N. For that, an empirical evaluation was performed based on the execution times of the *fast* and *neural* retrieval models. In theory, different values of N should not affect the *fast* retrieval models in terms of time efficiency.

In Table 6.5 is shown the average execution time per query over a total of 100 queries, that were registered for the following values of $N \in [1000, 2500, 5000]$. The execution time concerns the elapsed-time taken by the algorithm to compute the score for all the documents. The sorting time is the time taken to sort the scores of the retrieved documents. For the *neural* retrieval model, the sorting time was not noted because it was negligible, around 1ms.

Table 6.5: Record of empirical observations for different values of N

Models	Overall Average Time per query (seconds)					
	N=1000		N=2500		N=5000	
	Execution time	Sort time	Execution time	Sort time	Execution time	Sort time
BM25	2.27	1.81	2.31	1.86	2.29	1.91
AWE	0.68	0.73	0.67	0.73	0.68	0.73
AWE-TFIDF	0.67	0.75	0.68	0.75	0.68	0.76
DSSM	0.28	0.61	0.28	0.60	0.29	0.61
DeepRank	4.12	-	9.71	-	20.10	-
Self-Attn-DeepRank	3.94	-	9.52	-	18.86	-
HAR	1.93	-	5.09	-	9.33	-

As expected, the execution times of the *fast* retrieval were not affected by different N. However, unlike expected, the BM25[13] was the slowest of the *fast retrieval models*, which contradicts the intuition given by the complexity analysis. The explanation for this is that the BM25 executes one query at a time, while the others *fast* retrieval models use linear algebra, Equation 5.4, to compute the score for multiple queries at the same time. So the efficiency of these linear algebra computations improves their execution time. The fastest of the *fast* retrieval models was the DSSM model, due to a lower dimensionality (128) when compared to both AWE (200).

For the *neural* retrieval models, it is notable a linear penalization of the execution time in function of N, which reinforces the idea of including the filter module as a first step into the pipeline. Another interesting observation is that the *Self-Attn-DeepRank* has half of the parameters of the *DeepRank*. Therefore it was expected to be considerable faster. However,

[13]To be comparable with the others, the loading times of the inverted index were discarded, but as a side note they took approximately 50s each.

this is not verified, which indicates that the convolution layer must be the main reason for the higher execution time of the DeepRank model[14].

Continuing with the problem of finding an acceptable value of N, the next step is to evaluate, in terms of recall, the *fast* retrieval models for the same set of N values. Table 6.6 shows the recall values computed over the validation set of the training BioASQ data. For the BM25 and both AWE is shown the utilized tokenizer with respect to Table 6.2.

Table 6.6: Fast retrieval models results for different values of N in the validation set.

Fast Retrieval Models	Recall		
	N=1000	N=2500	N=5000
BM25 (tokenizer 1)	0.832	0.875	0.908
AWE (tokenizer 2)	0.251	0.322	0.390
AWE TF-IDF (tokenizer 2)	0.256	0.352	0.444
DSSM	0.041	0.061	0.077

As expected, the value of N is proportional to the value of the recall, which means that higher values of N will give a better chance to the *neural model* to get the best possible results, since it will score more positives documents. On the other hand, small values of N will give the possibility to train a higher number of models, which facilitate the fine-tuning process. Based on these considerations, when comparing the rate of growth of the execution times of the *neural model* and recall values of the *fast models* in function of N, it becomes clear that it may not be worth it to chose a bigger value for N, since the time penalization in the *neural model* would be too high. In the end, a compromise was made, an **N=2500 was the chosen value**.

Now that the value of N is established let us focus on choosing the best *fast retrieval model*. By looking at the results, it is clear that the BM25 was, by some margin, the best model, which reinforces the importance of the exact match signal, in retrieval tasks, between a query and document terms. On the other hand, the semantic match approaches fail to extract this type of exact signal and hence becoming not suitable for this specific task. In general, the results clearly show the difficulty of this retrieval task, recalling that the PubMed collection has almost than 19 millions of documents.

All the AWE variations had a similar performance, with a bit of gain to the TF-IDF variation. The overall results were worst than the BM25, this is easily explained by the negligence of the exact match signal. Another recurrent problem with this model is the number of words per document, that is a lot higher than the number of words per queries. So the document has more embeddings during the computation of their average, which can negatively affect the document final representation and harm the similarity computation. Another evidence that supports the previous analysis, it is the fact that the TF-IDF weighting system boosted the results, which means that the TF-IDF positively shifted the query representation (centroid) to be closer in terms of cosine similarity to the representation of the relevant documents.

[14]To verify this affirmation both models could be tested in GPU environment, which smooth the convolution times.

The DSSM had the worst results, which indicate that during training the model did not learn a meaningful way of projecting a query and a relevant document. The reason behind this observation can be related to the 128-dimensional space that is created to project both queries and documents. This dimension may be too small to condense all the information present in a document, which is also related with the input layer that uses bag-of-trigram to represent an entire document. Another problem is the higher number of trainable variables and the lack of training data when comparing it with the total number of documents in the collection.

Based on the analysis of the previous results, both AWE models were improved. In this new version, a greater emphasis was given to the less frequent words, supported by the idea that less frequent words are generally more important in retrieval tasks. The improvement consists in creating a new tokenizer, that will be designated as *"tokenizer 3"*, that only outputs a token if has a frequency less than 700000[15], i.e, if appears less than 700000 times in the data collection. So this tokenizer will not tokenize the most frequent words and the resulting vocabulary has 4291062 words. The previous recall test was performed to this new version and the results are shown in Table 6.7. In parenthesis is the relative improvement, in percentage, of the recall score with respect to its previous one.

Table 6.7: Improvements of the AWE models with the tokenizer 3 over the validation set.

Fast Retrieval Models	**Recall (improvement %)**		
	N=1000	**N=2500**	**N=5000**
AWE (tokenizer 3)	0.371 (47%)	0.470 (46%)	0.549 (41%)
AWE TF-IDF (tokenizer 3)	0.424 (66%)	0.529 (50%)	0.606 (36%)

In general, the AWE model receives a boost of almost 50%, which is extremely positive taking into consideration that the tokenizer 3 only has less 731 tokens when comparing with the tokenizer 2. Intuitively the boost seems to be related with the fact that this tokenizer produces shorter sentences while keeping the most important information.

Finally, the model that has the higher score given by Equation 5.1 is the chosen to power the *fast* retrieval module, this ensures that the model fulfil all the module requirements, defined in Section 5.2. For the computation of the score, it is needed to define the value of N and the maximum acceptable execution time T. N was already set to $N = 2500$ and it was defined that the maximum acceptable time will be five seconds, $T = 5$. So, Equation 5.1 becomes $s(d) = recall@2500$ because all the models times are inside the interval $]0, 5]$. Then the ideal model is the BM25 with a score of 0.875.

On a side note, the PubMed collection was also indexed by the *elastic search*[16] engine using the BM25 weights. This way it possible to validate the implementation of the BM25 and perhaps compare the performance. The *elastic search* was already running and configured in another machine, which means it was only needed to index the collection. Table 6.8 presents a comparison between the implemented BM25 version (1) and the *elastic search* version in

[15]This value was fine-tunned in the training set

[16]https://www.elastic.co/

terms of index times, search times and *recall*@2500 score. For the *elastic search* version, the times were recorded with (2) and without cache (3). From the results, the value of *recall*@2500 metric validate the implementation presented in this dissertation, note that the same BM25 hyper-parameters were used in both versions. In terms of times, it is shown that the implemented version is extremely competitive with the *elastic search* version without the caching mechanism[17]. At last, the average search time test is a bit biased, since it corresponds to the average of 100 queries and in this case, the time of an inverted index load in (1) are less severe because the index is reused for all the 100 queries, which *"hides"* the loading times. Although not recorded, if the inverted index for the version (1) was kept in memory the theoretical average search time will be $2.31 + 1.86 = 4.17s$[18], which are a lot higher than the version (3).

Table 6.8: Comparison between the implemented BM25 ranking function and *elastic search* BM25 version. The average search time is measure in a total of 100 queries.

BM25 Version	Indexing time	Average search time	RECALL@2500
Implemented BM25 (1)	02h18m28s	8.82s	0.875
Elastic Search (BM25) without cache (2)	02h17m37s	10.75s	0.875
Elastic Search (BM25) with cache (3)	-	0.39s	-

6.5 Neural Ranking Results

The neural models were trained, as a ranking task, over the 2500 previously retrieved documents for each query in the BioASQ training set. Then their efficacy is evaluated in terms of MAP@10 and *recall*@10 using the BioASQ validation set. Following Equation 5.5, the best neural model will be the model that has a higher value of MAP@10.

In the following subsections, it is analysed the training behaviour of each neural model. For that, it is presented an evolution of the MAP score and loss value during the training iterations (epochs). The MAP value was only computed on a subset of the validation set to mitigate the higher inference times. This subset corresponds to just 15% of the original validation set resulting in a total of 82 queries, which are enough to give an idea of the model capability for generalizing to new data. Additionally, the MAP score is measure on intervals of 10 epochs since even with only 82 queries, it would be impractical to measure in every epoch due to time limitations. For the loss value, it is presented the maximum, average, and minimum value for each epoch on the training set. Also note that the validation loss is not presented since it does not provide useful information about the model performance. This affirmation is sustained by the fact that the model performance should be measured by its final ranking order, however, the hinge loss only measures a relative preference between two documents to the same query.

As a way to clarify the terminology, in this work, an epoch is considered as a complete run over the training set, since the data is fed in batches that are smaller than the training set,

[17]This mechanism also includes the caching of the FileSystem
[18]This are the recorded average values presented on Table 6.5

each epoch will correspond to several steps with a fixed batch size. The training of each neural model was done in an interactive way, e.g, initially, the model could be trained only on 50 epochs, then based on the results more epochs were added until it converges to a satisfactory minimum in terms of loss. The number of samples per batch was chosen based on the trade-off of efficiency of training and the noise of the gradients. Very briefly, a big batch size severely increases the training time and the resources needed, while with the small batches, the model will be more susceptible to bad weight updates due to poor data.

6.5.1 DeepRank training behaviour

As presented in Table 6.9, the DeepRank model was trained over a total of 230 epochs, where each epoch has 8 steps with a batch size of 256, i.e, the model weights are updated $230 \times 8 = 1840$ times.

Table 6.9: Additional information of the training hyperparameters for the DeepRank model

Additional training information	
Epoch	230
Step per epoch	8
Batch size per step	256
Total training time (h)	3h48m57s
Total evaluation time (h)	5h05m13s

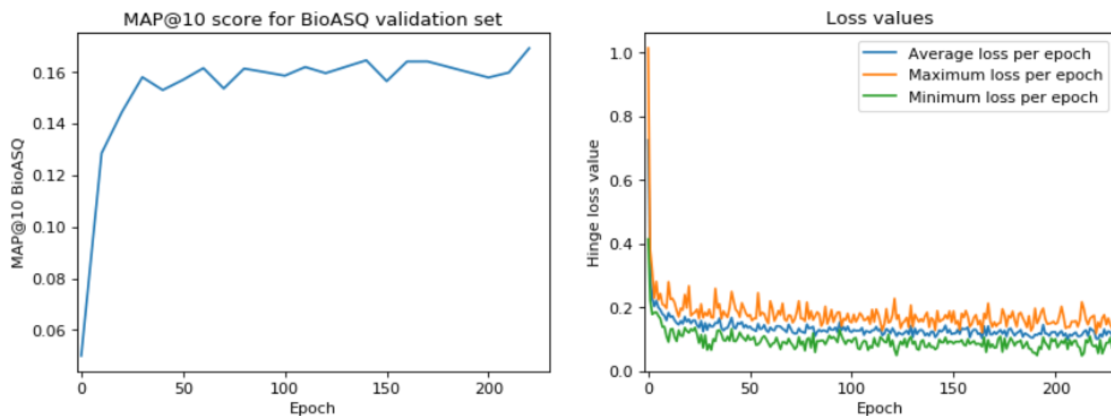


Figure 6.4: On the left, it is the evolution of the map score in the subset of the validation set and on the right, the evolution of the loss during training for the DeepRank model.

From Figure 6.4, it is possible to see that the model quickly converges after 30 epochs, in both validation and loss graph, then the values seem to stabilize. Additionally, more advanced techniques could also be applied, like learning rate decay, to try to get a better minimum. However, in this work, this was not adopted because it will difficult the comparisons between the models since it introduces more variation between the models.

An important note to take into consideration during the loss analysis, it is that the average value of the loss corresponds to 8 values from each step of an epoch, so for the same epoch the final step value of the loss is 8 iterations ahead of the first one, since the model already

performed 8 weight updates. This also explains the fact that in the epoch 1 the maximum and minimum values are so separated.

6.5.2 Self-Attn-DeepRank training behaviour

The same training hyperparameters were chosen to this model, as shown in Table 6.10, which also facilitates the comparations between the two models. In terms of total time, this model was marginally faster than the previous one, once again, the convolution operation should be the operation that takes longer to compute, which explains the proximity in terms of times.

Table 6.10: Additional information of the training hyperparameters for the Self-Attn-DeepRank model

Additional training information	
Epoch	230
Step per epoch	8
Batch size per step	256
Total training time (h)	3h42m16s
Total evaluation time (h)	4h59m14s

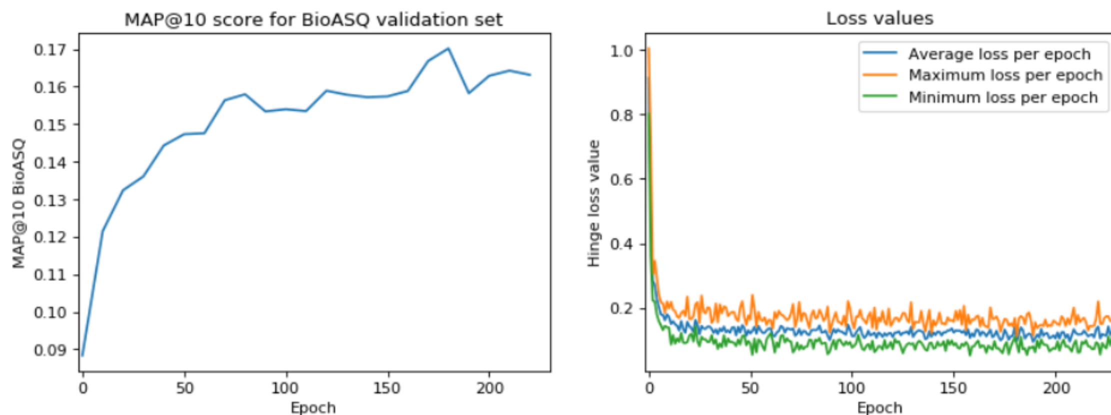


Figure 6.5: On the left, it is the evolution of the map score in the subset of the validation set and on the right, the evolution of the loss during training for the Self-Attn-DeepRank model.

In Figure 6.5 and also comparing with the loss graph of the previous model, it seems that this model converges a bit faster in terms of loss function, just after 20 epochs. However, in terms of performance takes longer to stabilize, this behaviour was unexpected but could be related with some variance from the negative sampling.

6.5.3 HAR training behaviour

For the HAR model, the 230 epochs were not enough based on the loss values Figure 6.6, so more epoch was added in an incremental way making a total of 900 epoch. The same batch size was used, which means that during training this model performed a total of $900 \times 8 = 7200$ weight updates. Due to a considerably higher number of iterations, the total training and evaluation times were also a lot higher.

From the loss graph, on the right, it seems that the model could be trained for more iterations. However as previously mentioned, this model is more prone to overfit so the

Table 6.11: Additional information of the training hyperparameters for the Hierarchical Attention Retrieval model

Additional training information	
Epoch	900
Step per epoch	8
Batch size per step	256
Total training time (h)	8h19m12s
Total evaluation time (h)	10h26m04s

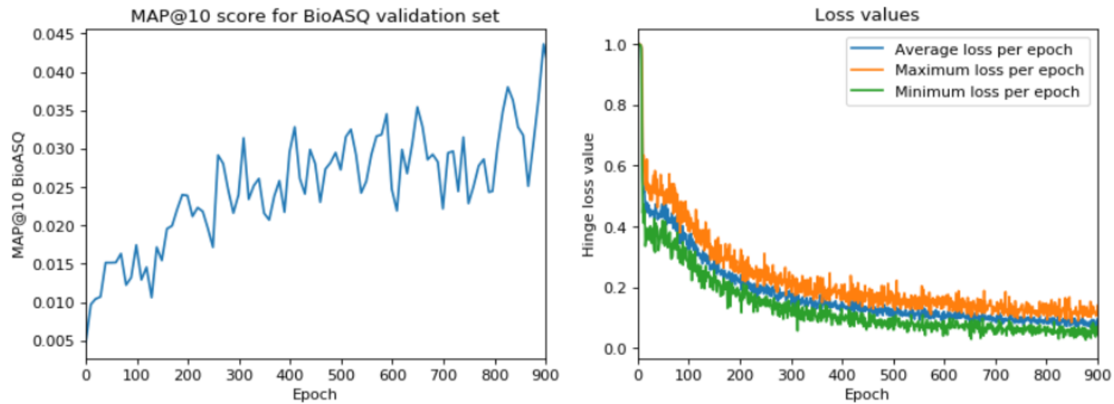


Figure 6.6: Graphic.

presented loss behaviour can be an indicator of overfitting as opposed to *lake of training iterations*. On the other hand, the poor results on the validation set also support the idea of overfitting, since it was not capable of generalizing to new data. But this will be discussed more ahead.

6.5.4 Neural models comparison

In this subsection, the best neural models[19] that were obtained during training will be evaluated on the complete validation set. This evaluation uses the MAP@10 and *recall*@10 metrics and the results are presented in Table 6.12. Additionally, the BM25 was also included in the results table to verify if, indeed, the neural retrieval models were capable of improving its ranking order. The publicly available BioASQ-Evaluation[20] Code, written in java, was used to compute the MAP and *recall* values. This way is ensured that the best model is chosen based on the BioASQ metrics, which uses a more penalize version of MAP, Equation 6.1.

From the results, both variations of the DeepRank model achieved an improvement of 10% over the previous ranking order (BM25 result), which, curiously, it is also the same percentage of improvement that the DeepRank authors got on the MQ2007 dataset. It should also be noted that the *Self-Attn-Deeprank*, which has half of the training parameters of the *DeepRank* were capable of achieving the same performance.

[19]The best model during train, it corresponds to the model that has the higher MAP score.
[20]https://github.com/BioASQ/Evaluation-Measures/tree/master/flat/BioASQEvaluation

Table 6.12: Evaluation of the neural retrieval models in the complete validation set

Models	MAP@10	RECALL@10
BM25 (baseline)	0.153	0.330
DeepRank	0.168 (10%)	0.356 (8%)
Self-Attn-DeepRank	**0.168 (10%)**	**0.358 (9%)**
HAR	0.036	0.100

As suspected, the HAR model performs poorly, possibly due to the lack of training data. To confirm this hypothesis, the same metrics were performed using this time the training set. The results were $MAP@10 = 0.176$ and $recall@10 = 0.520$, which clearly confirms the overfitting of the training data. Due to these results, this model will be discarded for the following analysis.

As suggested in the literature [56], a usual way to compare different retrieval models is to analyse the precision-recall curves. In practice, for each model, the precision of a model is plotted in function of 11 recall levels $\{0, 0.1, ..., 1\}$. Normally the precision must be interpolated, since it may not be possible to obtain the exact value for a given recall level, Equation 6.2.

$$P_{interpolated}(r) = max_{r' \geq r}(Precision(r')) \tag{6.2}$$

Here, for a given recall level (r) the maximum value of precision found in the interval $[r, 1]$ is returned.

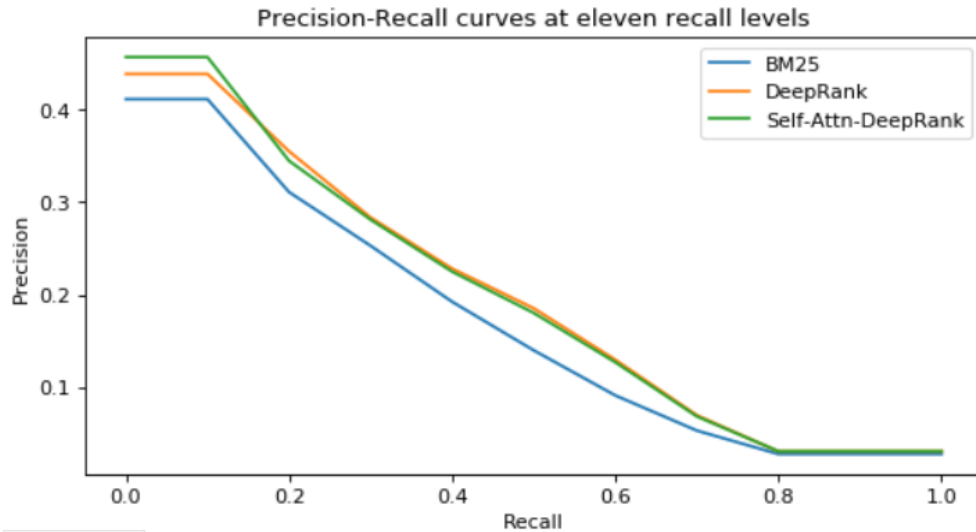


Figure 6.7: Graphic with precision-recall curves at eleven recall levels for both DeepRank variations and BM25. The precision was computed over the 2500 documents that correspond to the complete ranking order produced by both neural models.

In Figure 6.7 are presented the curves for the DeepRank model, in yellow, the Self-Attn-DeepRank, in green, and the BM25, in blue. As shown both lines are over the baseline (BM25), which clearly indicates a superior performance of both models. Another interesting

fact, it is that the Self-Attn-DeepRank seems to have a significantly better precision at low recall levels when compared to the DeepRank.

A further investigation was performed to analyze the re-ranking behavior of both models. For that, the MAP@N value was computed in function of different values of N.

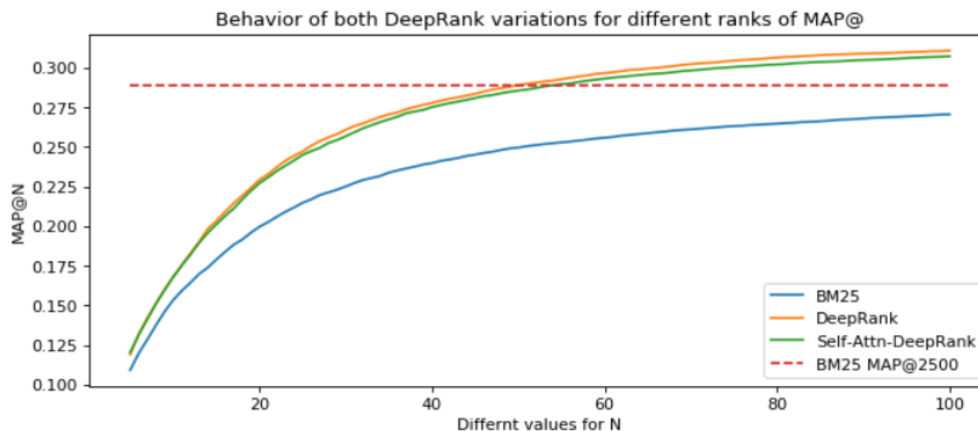


Figure 6.8: Graphic of MAP@N in function of different N for both DeepRank variations.

Figure 6.8, shows the results of this additional experiment and it can be observed an increasing gap between the DeepRank variations and the BM25 (before the re-rank). This shows that the DeepRank is correctly scoring (ranking) relevant documents that were overlooked by the BM25, i.e, the false negatives documents of the BM25 are being pushed to the top by the DeepRank. Additionally, at $N = 50$ the MAP value already surpasses the best MAP value of the BM25, which demonstrates the power of the neural solution.

In general, taking into consideration the small dataset of the BioASQ, the results of both DeepRank models were extremely positives and both were chosen to be included in the pipeline so that can be tested. It is worth to recall, that the both DeepRank models results are limited by the previous BM25 results since if the BM25 fail to retrieve any positive document, the DeepRank will not be able to score it.

6.6 Snippet Extraction Results - visualization

The main idea of this module is to offer a way of visualizing, according to the neural model perspective, the most important information inside a document with respect to a given query. So this section presents an illustrative example of this type of visualization followed by a respective analysis.

Proceeding with the visualization, it is shown a positive and a negative document for the same query[21], the positive document was ranked in the first position at the ranking list and the negative document is the first false positive document in the ranking list, corresponding to the third position. Figure 6.9 shows the query-positive pair and Figure 6.10 shows the query-negative pair for the query *"Which enzyme is inhibited by Imetelstat?"*. Both visualizations

[21]The chosen example was the best from a set of randomly selected query-document pairs

Tokenized query

enzyme inhibited imetelstat

Tokenized document

the telomerase inhibitor imetelstat alone and in combination with trastuzumab decreases the cancer stem cell population and self renewal of her2 breast cancer cells cancer stem cells cscs are thought to be responsible for tumor progression metastasis and recurrence her2 overexpression is associated with increased cscs which may explain the aggressive phenotype and increased likelihood of recurrence for her2 breast cancers telomerase is reactivated in tumor cells including cscs but has limited activity in normal tissues providing potential for telomerase inhibition in anti cancer therapy the purpose of this study was to investigate the effects of a telomerase antagonistic oligonucleotide imetelstat grn163l on csc and non csc populations of her2 breast cancer cell lines the effects of imetelstat on csc populations of her2 breast cancer cells were measured by aldh activity and cd44 24 expression by flow cytometry as well as mammosphere assays for functionality combination studies in vitro and in vivo were utilized to test for synergism between imetelstat and trastuzumab imetelstat inhibited telomerase activity in both subpopulations moreover imetelstat alone and in combination with trastuzumab reduced the csc fraction and inhibited csc functional ability as shown by decreased mammosphere counts and invasive potential tumor growth rate was slower in combination treated mice compared to either drug alone additionally there was a trend toward decreased csc marker expression in imetelstat treated xenograft cells compared to vehicle control furthermore the observed decrease in csc marker expression occurred prior to and after telomere shortening suggesting that imetelstat acts on the csc subpopulation in telomere length dependent and independent mechanisms our study suggests addition of imetelstat to trastuzumab may enhance the effects of her2 inhibition therapy especially in the csc population

Figure 6.9: Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a true positive document at position 1 of the ranked list.

Tokenized query

enzyme inhibited imetelstat

Tokenized document

telomerase inhibition abolishes the tumorigenicity of pediatric ependymoma tumor initiating cells pediatric ependymomas are highly recurrent tumors resistant to conventional chemotherapy telomerase a ribonucleoprotein critical in permitting limitless replication has been found to be critically important for the maintenance of tumor initiating cells tics these tics are chemoresistant repopulate the tumor from which they are identified and are drivers of recurrence in numerous cancers in this study telomerase enzymatic activity was directly measured and inhibited to assess the therapeutic potential of targeting telomerase telomerase repeat amplification protocol trap n 36 and c circle assay telomere fish atrx staining n 76 were performed on primary ependymomas to determine the prevalence and prognostic potential of telomerase activity or alternative lengthening of telomeres alt as telomere maintenance mechanisms respectively imetelstat a phase 2 telomerase inhibitor was used to elucidate the effect of telomerase inhibition on proliferation and tumorigenicity in established cell lines bxd 1425epn r254 a primary tic line e520 and xenograft models of pediatric ependymoma over 60 of pediatric ependymomas were found to rely on telomerase activity to maintain telomeres while no ependymomas showed evidence of alt children with telomerase active tumors had reduced 5 year progression free survival 29 11 vs 64 18 p 0 03 and overall survival 58 12 vs 83 15 p 0 05 rates compared to those with tumors lacking telomerase activity imetelstat inhibited proliferation and self renewal by shortening telomeres and inducing senescence in vitro in vivo imetelstat significantly reduced subcutaneous xenograft growth by 40 p 0 03 and completely abolished the tumorigenicity of pediatric ependymoma tics in an orthotopic xenograft model telomerase inhibition represents a promising therapeutic approach for telomerase active pediatric ependymomas found to characterize high risk ependymomas

Figure 6.10: Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a false negative document that appears at position 3 of the ranked list.

were performed with the *Self-Attn-DeepRank* model and the maximum number of returned snippets is fixed to ten.

In both visualization, at the top is presented the tokenized query and for each query token, the respective model's attention is highlighted with a red background. The color intensity indicates the importance that is given to each token, where more important tokens are more darker. At the bottom is presented the document after the tokenization, where each snippet is highlighted in blue, with a variable color intensity according to its importance with respect to the model perspective.

From the examples, the model considers the *Imetelstat* as the most important word in the query followed by the term *enzyme*, which it seems to be a plausible distribution of attention because it respects the intuition that rare terms tend to be more important. Note that the notion of rare terms was not fed to the model and it is something that the model was capable of learning.

It is a bit difficult to judge if the highlighted snippets are indeed correct, due to the lake of

knowledge in the biomedical domain. However, in the positive document, from the highlighted snippets it possible to infer that *telomerase* should be an enzyme that is inhibited by the *imetelstat*, that indeed is true. Following this assumption, the presented negative document should be a positive document that was misclassified by the experts, since it is possible to make the same inference from the highlighted snippets. More examples of different queries can be found in Appendix A.

Tokenized query

enzyme inhibited imetelstat

Tokenized document

characterization of a novel carboxypeptidase produced by the entomopathogenic fungus metarhizium anisopliae preparative isoelectric focusing and gel filtration chromatography were used to purify a carboxypeptidase produced by the entomopathogenic fungus metarhizium anisopliae during growth on cockroach cuticle the enzyme was inhibited by diisopropyl fluorophosphate implying involvement of a serine residue in catalysis however the m anisopliae enzyme differed from most serine carboxypeptidases in also being inhibited by the metal chelator 1 10 phenanthroline and in being a small 30 kda basic pi 9 97 protein with a neutral ph optima ph 6 8 these properties resemble those exhibited by some metalloproteases but the enzyme is not inhibited by cd2 nor do zn2 or co2 restore activity in enzyme inhibited with phenanthroline the amino terminal sequence 22 residues showed no similarity to other protein sequences unlike previously reported fungal carboxypeptidases the m anisopliae enzyme is powerfully inhibited by potato carboxypeptidase inhibitor the carboxypeptidase shows a broad primary specificity toward amino acids with hydrophobic side groups in a series of n blocked dipeptides with substrates with phenylalanine being the most rapidly hydrolyzed the s1 subsite also accommodated glu confirming its low selectivity proline at p1 or p1 resulted in a very poor substrate the specificity of the carboxypeptidase complements that of the subtilisin like protease pr1 of m anisopliae both pr1 and the carboxypeptidase are produced during carbon and nitrogen deprivation which indicates that the exopeptidase functions with pr1 to degrade peptides to supply amino acids during starvation and pathogenicity

Figure 6.11: Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a negative document that it is ranked at position 50.

In Figure 6.11 is presented another negative document but this time with a lower score, more precisely this document only appears at position 50 in the ranking list. As can be seen, the color of the snippets highlights is more lighter giving an idea that information of this snippets is less important with respect to the query terms.

So in general, the snippet highlighting provided by the attention levels of the neural model seems to be a useful mechanism that can be exploited to get a correct answer. However, this is something that can be further investigated in a more methodical way in the future. Additionally, this mechanism also helps to analyse/debug the neural model behaviour by looking at the attention levels.

6.6.1 DeepRank - inner working analysis

This subsection presents a more speculative analysis of the DeepRank models based on their behaviour and results. The goal is to explain why the DeepRank is capable of improving the BM25 ranking order and what operations contribute the most for that.

In the first place, the aggregation network seems to not have a major role in the DeepRank performance. This affirmation is sustained on the fact that both variations (DeepRank and Self-Attn-DeepRank) had similar performance with different types of aggregation networks. Additionally, the boost in performance that the DeepRank is capable of achieving, with respect to the BM25, must be related with its wider context view over the query and snippet input, i.e, the BM25 only performs an exact term matching search, while the DeepRank takes into consideration the context of where this match occurs, giving more information that probably helps to score this match. As an example let us explore a slice of one of the interaction

matrices that is built for the previously presented positive query-document pair Figure 6.9, for sake of simplicity let us also considered only a subset of the snippet terms (*... imetelstat inhibited telomerase ...*)[22] and the query terms (*enzyme inhibited imetelstat*), the resulting slice of the matrix is shown in Table 6.13.

Table 6.13: Slice of the expected interaction matrix for query terms *enzyme inhibited imetelstat* and snippet terms *imetelstat inhibited telomerase*.

	imetelstat	inhibited	telomerase
enzyme	low	low	high
inhibited	low	1	low
imetelstat	1	low	low

Here, the high and low values belong to the cosine domain, i.e, the high value should be a value close to 1 and a low should be close to -1[23]. The matrix entries with value 1 corresponds to the maximum similarity value that occurs for the exact matches. Continuing with the example, the *enzyme-telomerase* interaction is expected to have a higher value of similarity, because the term *telomerase* is indeed an *enzyme* and this relation should be presented in their embeddings representation, so a high similarity value is captured by the normalized dot product operation when the matrix is built. Then a convolutional filter, that has a window 3 by 3, when applied to this matrix should give a high score to this type of pattern, that is then extracted by the max pooling layer, i.e, the presented matching signal is successfully extracted by this sequence of operations. Assuming that these suppositions are true, it can be concluded that this architecture represents a *context-aware term matching extraction process*, which seems to have a major role in the final document score since it represents the core process of a retrieval task. On the other hand, the aggregation network and the term gating network are only a weighting mechanism to refine the matching signal previously extracted.

It is also worth to mention, that the attention that is given to the query terms, in the gating network, is only based on a linear combination between a weight matrix and the respective query terms embedding. As a result, the learned matrix must be capable of successfully extract the information from the embeddings dimensions to compute its importance. The most simplistic example could be that one of the embeddings dimensions is responsible to encode the relative term importance, which seems plausible given that they are created by the *word2vec* algorithm. However, in practice, it may not be that simple, nerveless a more complete analysis of this learned weight matrix can give more insights about some of the embeddings dimensions and help to understand the distributed representation that is built during the *word2vec* training.

6.7 BioASQ 7 taskB phaseA results

In this section is presented the evaluation of the overall system on the 2019 BioASQ 7b phaseA test set for the document retrieval task. The evaluation is performed locally using the BioASQ

[22] First highlighted snippet on Figure 6.9

[23] In practice, this may not be true, however at least it will be a lower value when compared to the *enzyme-telomerase* similarity value

metrics and the 2019 BioASQ 7b phaseB test set, which has the relevant documents for the phaseA.

In terms of testing, were prepared two systems, since both DeepRank based models got a similar performance. System 1 will be referred as the system that uses the original *DeepRank* and System 2 will be the system that uses the *Self-Attn-DeepRank* as the neural model. For the filter module, both systems use the BM25.

Table 6.14: Comparison with other submissions of the bioASQ task 7b phaseA results

System	Test Batch 1			Test Batch 2			Test Batch 3		
	MAP@10	S Pos	G Pos	MAP@10	S Pos	G Pos	MAP@10	S Pos	G Pos
Best result	0.0809	-	-	0.0849	-	-	0.1199	-	-
System v1	**0.0874**	**1/12**	**1/6**	0.0760	7/23	4/9	0.1006	6/21	3/8
System v2	0.0865	1/12	1/6	0.0764	7/23	4/9	0.0995	6/21	3/8

System	Test Batch 4			Test Batch 5		
	MAP@10	S Pos	G Pos	MAP@10	S Pos	G Pos
Best result	0.1034	-	-	0.0425	-	-
System v1	0.0922	5/17	2/6	0.0344	9/18	3/6
System v2	0.0882	6/17	2/6	**0.0373**	**3/18**	**2/6**

Table 6.14 compares both systems with other submitted systems, available online[24], in all of the five released batches. The complete tables are provided in Appendix D. Column *"S Pos"* (system position) represents the absolute position that the system would have if it had been submitted, i.e, (x/y) x is the system position and y is the total number of valid submissions. In the same way, the column *"G Pos"* (group position) represents the group position, since each researcher (group) is allowed to submit a maximum of five systems. The line *Best result* corresponds to the result of the best submitted system for each test batch.

In general, the results were extremely competitive and both systems were able to achieve satisfactory results. For example, in Batch 1 both systems achieve, with some margin, better results than the best submitted systems. For the Batch 2,3 and 4, both systems stayed close to the top with competitive map scores. In the final Batch (5) the System v2 achieved the third best result.

It is worth mention, that after the Batch 1 the group *aueb*[25] submitted five systems that got almost every top spot on Batch 2 to 5. So the second place on the Batch 5, has an even bigger importance since it means that System v2 was capable of beat three of the five systems of *aueb* group. To give more context the *aueb* system [68], [84], partially presented in Section 4 as a DRMM improvement, in the last year competition comes in first place on three of the five test Batch and in their team is a researcher that works at Google AI[26].

6.8 System as web application

To conclude this dissertation work, it will be presented the interaction with the web application as a set of sequential images.

[24]http://participants-area.bioasq.org/results/7b/phaseA/ (30/06/2016)
[25]http://nlp.cs.aueb.gr/
[26]https://ai.google/

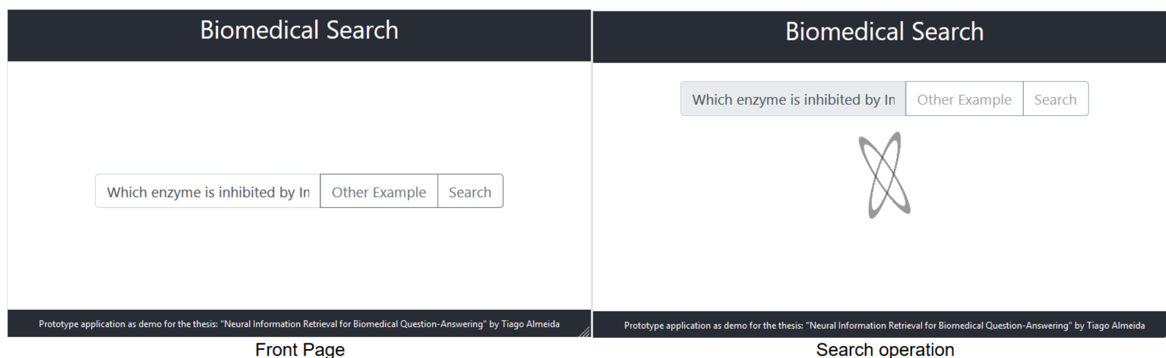


Figure 6.12: Front page of the application on the left and search request on the right. Both images where captured with resolution of 640×360.

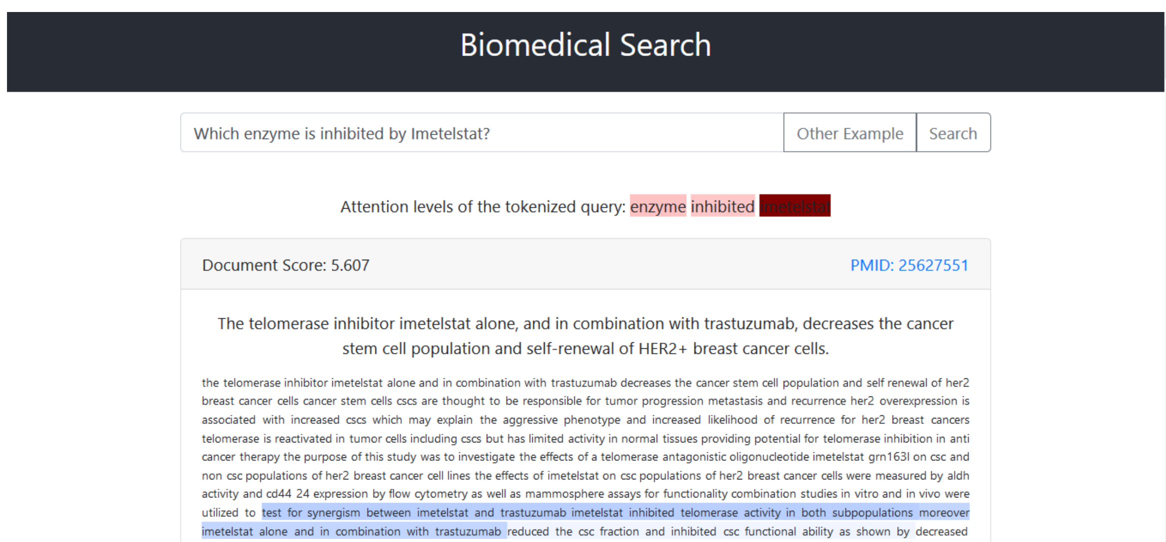


Figure 6.13: Display of the results for the previous search. The image was capture with resolution of 1366×768.

In Figure 6.12, on the left, it is displayed the front page with an example query from the BioASQ dataset, other queries can be load by using the "Other Example" button. On the right, it is shown the state of the web application after a request was been submitted.

The API response time was also evaluated for a total of ten identical requests, the average waiting time was 16.66 seconds. This value was expected, since the neural model on its own takes approximately 10 seconds, according to Table 6.5. The additional six seconds must be related with the BM25, data preparation and *http* overhead. In practice, from a user point of view, the 16 seconds as waiting time is acceptable given the quality of the results and also recalling that this application is just a prototype that can be improved.

CHAPTER 7

Conclusion and Future work

With the growth of the publicly available biomedical literature, this dissertation aimed to create a deep learning based system capable of searching a vast biomedical literature given a natural language question. Additionally, it also gives some form of intuition that should be followed in order to keep pushing the research forward in the area of neural information retrieval in the biomedical domain.

Initially, this document presents a background about neural networks, advanced deep learning techniques and an introduction to the information retrieval. The purpose is to give a theoretical support to the state-of-the-art models that are explored and implemented during this dissertation.

Then, in Chapter 5 it is shown the pipeline of question-answering system. In short, the system is composed of the three following modules, filter, rank, and extract. The filter module will search the vast literature and return the top 2500 documents that are plausible relevant to the query. For this module were tested a retrieval model focus in exact match, the *bm25*, and several models focused in semantic match, the AWE, AWE-TF-IDF, and DSSM. The rank module uses deep neural models to score all the plausible relevant documents jointly with the query, with the objective to give higher scores to relevant documents. Additionally, only the deep neural models that directly used a set of snippets as a document representation were considered. This restriction is based on the premise that a high score for each document is directly correlated with their individual snippets. The DeepRank model and the HAR model were the only deep neural models that respect this requirement. Finally, the extracting module will highlight which of the snippets, from the documents, are relevant. This last module uses the deep model from the previous step, to directly extract this information from its activations.

The evaluation, presented in Chapter 6, was conducted in a systematic and methodical way. In first place, the requirements that each module should achieve were tested by their respective evaluation function. In the case of the filter module, the objective was to find a commitment between the maximization of the recall and the minimization of the number of

returned documents. On the rank module the objective was to maximize the map@10 metric. Looking at the results for the first module, Table 6.6, the *bm25* got the overall best results, showing that when searching a vast collection with sparse vocabulary the exact match model had an upper-hand. The AWE results were expected given the small embedding dimension when compared with the vast search space and the length difference of documents and queries. At last, DSSM had extremely low results, the main reason may be related with the insufficient training data and the small dimension (128) of the query/document projection. For the rank module, both DeepRank variants had satisfactory results on the validation set, with an improvement of **10%** over the BM25 ranking order. Also as suspected the HAR was not capable to generalize for new data. At last, for the snippet extraction module is only presented a visualization, which seems to support the evidence that the model score is depended only on a subset of relevant snippets. It was also developed a web application that exposes, in an interactive way, this system pipeline.

The overall system was evaluated on BioASQ 7 task B phase A, where it obtained the **best score** on first batch and the **third best** score on the last batch, when compared to the submitted models.

The final system has some known limitation, one that can be pointed out is the elapsed time needed to computed a single query. However, all the retrieval models are prototypes, so they can be considerable speed up. Another limitation has the lack of time to do a more thorough search to the model's hyperparameters. In terms of DeepRank, the concept of snippet could be enhanced to build better representations since for now is using a fixed size window.

This dissertation supports the following evidence present through the literature: "The exact match signal must be taking into consideration with higher weight than the semantic match for the specific retrieval task". It is also shown the implementation of two state-of-the-art deep neural models, which at time of writing do not exist any publicly available solution.

For future work, exist the possibility of further evolving the current system or add a text generator to the system.

To continue the evolution of this system the following ideas can be adopted.

- Incorporate the *bm25* jointly with some weighted semantic retrieval system, for example, the AWE-TF-IDF. Note that a bigger weight should be given to the *bm25*.
- Directly improve de DeepRank models results by doing a better fine-tunning and additionally exploring the possibility of using ensemble of multiple trained version of the same neural model.
- Unify the rank and the extract module, so that during training the deep neural model take directly into consideration the snippets that are more important. This can be achieved by using a document loss plus a weighted snippet loss.

The text generation path will consist in creating a human-readable sentence, given a query and a document or set of snippets that are relevant, which means that the text generator must be conditioned on relevant information. In terms of implementations, the state-of-the-art *Transformer Block* can be explored in order to generate high quality sentences.

References

[1] "Detailed Indexing Statistics: 1965-2017", [Online]. Available: https://www.nlm.nih.gov/bsd/index_stats_comp.html.

[2] *Home - PubMed - NCBI*. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/ (visited on 12/31/2018).

[3] P. Malakasiotis, I. Androutsopoulos, A. Bernadou, N. Chatzidiakou, E. Papaki, P. Constantopoulos, I. Pavlopoulos, A. Krithara, Y. Almyrantis, D. Polychronopoulos, A. Kosmopoulos, G. Balikas, I. Partalas, G. Tsatsaronis, and N. Heino, "Challenge Evaluation Report 2 and Roadmap", Tech. Rep., 2014, p. 90. [Online]. Available: http://www.bioasq.org/sites/default/files/PublicDocuments/BioASQ_D5.4-Challenge_Evaluation_Report_2_and_Roadmap_final.pdf.

[4] L. HIRSCHMAN and R. GAIZAUSKAS, "Natural language question answering: the view from here", *Natural Language Engineering*, vol. 7, no. 04, pp. 275–300, Dec. 2001, ISSN: 1351-3249. DOI: 10.1017/S1351324901002807. [Online]. Available: http://www.journals.cambridge.org/abstract_S1351324901002807.

[5] *Challenge Overview | bioasq.org*. [Online]. Available: http://bioasq.org/participate/challenges (visited on 10/17/2018).

[6] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A Fast Learning Algorithm for Deep Belief Nets", *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006, ISSN: 0899-7667. DOI: 10.1162/neco.2006.18.7.1527. [Online]. Available: http://www.mitpressjournals.org/doi/10.1162/neco.2006.18.7.1527.

[7] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, Dec. 1943, ISSN: 0007-4985. DOI: 10.1007/BF02478259. [Online]. Available: http://link.springer.com/10.1007/BF02478259.

[8] A. Gron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st. O'Reilly Media, Inc., 2017, ISBN: 1491962291, 9781491962299.

[9] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization", *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Jul. 2011, ISSN: 1532-4435. [Online]. Available: http://dl.acm.org/citation.cfm?id=1953048.2021068.

[10] M. D. Zeiler, "ADADELTA: an adaptive learning rate method", *CoRR*, vol. abs/1212.5701, 2012. arXiv: 1212.5701. [Online]. Available: http://arxiv.org/abs/1212.5701.

[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization.", *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1412.html#KingmaB14.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research", in, J. A. Anderson and E. Rosenfeld, Eds., Cambridge, MA, USA: MIT Press, 1988, ch. Learning Representations by Back-propagating Errors, pp. 696–699, ISBN: 0-262-01097-6. [Online]. Available: http://dl.acm.org/citation.cfm?id=65669.104451.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014, ISSN: 1532-4435. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627435.2670313.

[14] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", *CoRR*, vol. abs/1207.0580, 2012. arXiv: 1207.0580. [Online]. Available: http://arxiv.org/abs/1207.0580.

[15] C. Angermueller, T. Pärnamaa, L. Parts, and O. Stegle, "Deep learning for computational biology", *Molecular Systems Biology*, vol. 12, no. 7, p. 878, 2016. DOI: 10.15252/msb.20156651. eprint: https://www.embopress.org/doi/pdf/10.15252/msb.20156651. [Online]. Available: https://www.embopress.org/doi/abs/10.15252/msb.20156651.

[16] S. Saha, "A comprehensive guide to convolutional neural networks", *Towards Data Science*, Dec. 2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257.

[18] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions", in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2015, pp. 1–9, ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298594. [Online]. Available: http://ieeexplore.ieee.org/document/7298594/.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2016, pp. 770–778, ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90. [Online]. Available: http://ieeexplore.ieee.org/document/7780459/.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *CoRR*, vol. abs/1409.1556, 2014. arXiv: 1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556.

[21] Z. Yang, T. Dan, and Y. Yang, "Multi-temporal remote sensing image registration using deep convolutional features", *IEEE Access*, vol. PP, pp. 1–1, Jul. 2018. DOI: 10.1109/ACCESS.2018.2853100.

[22] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences", *CoRR*, vol. abs/1404.2188, 2014. arXiv: 1404.2188. [Online]. Available: http://arxiv.org/abs/1404.2188.

[23] Y. Kim, "Convolutional neural networks for sentence classification", *CoRR*, vol. abs/1408.5882, 2014. arXiv: 1408.5882. [Online]. Available: http://arxiv.org/abs/1408.5882.

[24] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh and M. Titterington, Eds., ser. Proceedings of Machine Learning Research, vol. 9, Chia Laguna Resort, Sardinia, Italy: PMLR, May 2010, pp. 249–256. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html.

[25] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks", in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML'13, Atlanta, GA, USA: JMLR.org, 2013, pp. III-1310–III-1318. [Online]. Available: http://dl.acm.org/citation.cfm?id=3042817.3043083.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[27] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization", *CoRR*, vol. abs/1409.2329, 2014. arXiv: 1409.2329. [Online]. Available: http://arxiv.org/abs/1409.2329.

[28] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation", *CoRR*, vol. abs/1406.1078, 2014. arXiv: 1406.1078. [Online]. Available: http://arxiv.org/abs/1406.1078.

[29] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017, ISSN: 2162-237X. DOI: 10.1109/TNNLS.2016.2582924.

[30] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks", in *Proc. NIPS*, Montreal, CA, 2014. [Online]. Available: http://arxiv.org/abs/1409.3215.

[31] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate", *CoRR*, vol. abs/1409.0473, 2014. arXiv: 1409.0473. [Online]. Available: http://arxiv.org/abs/1409.0473.

[32] M. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation", *CoRR*, vol. abs/1508.04025, 2015. arXiv: 1508.04025. [Online]. Available: http://arxiv.org/abs/1508.04025.

[33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need", *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762.

[34] Z. Lin, M. Feng, C. N. dos Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A structured self-attentive sentence embedding", *CoRR*, vol. abs/1703.03130, 2017. arXiv: 1703.03130. [Online]. Available: http://arxiv.org/abs/1703.03130.

[35] C. D. Manning, P. Raghavan, and H. Schütze, "Boolean retrieval", in *Introduction to Information Retrieval.* Cambridge University Press, 2008, pp. 1–17. DOI: 10.1017/CBO9780511809071.002.

[36] K. D. Onal, Y. Zhang, I. S. Altingovde, M. M. Rahman, P. Karagoz, A. Braylan, B. Dang, H. L. Chang, H. Kim, Q. McNamara, A. Angert, E. Banner, V. Khetan, T. McDonnell, A. T. Nguyen, D. Xu, B. C. Wallace, M. de Rijke, and M. Lease, "Neural information retrieval: at the end of the early years", *Information Retrieval Journal*, vol. 21, no. 2-3, pp. 111–182, Jun. 2018, ISSN: 15737659. DOI: 10.1007/s10791-017-9321-y. [Online]. Available: http://link.springer.com/10.1007/s10791-017-9321-y.

[37] C. D. Manning, P. Raghavan, and H. Schütze, "The term vocabulary and postings lists", in *Introduction to Information Retrieval.* Cambridge University Press, 2008, pp. 18–44. DOI: 10.1017/CBO9780511809071.003.

[38] ——, *Introduction to Information Retrieval.* New York, NY, USA: Cambridge University Press, 2008, ISBN: 0521865719, 9780521865715.

[39] ——, "Scoring, term weighting, and the vector space model", in *Introduction to Information Retrieval.* Cambridge University Press, 2008, pp. 100–123. DOI: 10.1017/CBO9780511809071.007.

[40] S. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford, "Okapi at trec-3", in *Overview of the Third Text REtrieval Conference (TREC–3)*, Gaithersburg, MD: NIST, Jan. 1995, pp. 109–126. [Online]. Available: https://www.microsoft.com/en-us/research/publication/okapi-at-trec-3/.

[41] C. D. Manning, P. Raghavan, and H. Schütze, "Okapi bm25: A non-binary mode", in *Introduction to Information Retrieval.* Cambridge University Press, 2008, pp. 232–234. DOI: 10.1017/CBO9780511809071.007.

[42] B. Mitra and N. Craswell, "An Introduction to Neural Information Retrieval", *Foundations and Trends® in Information Retrieval*, pp. 1–117, 2018. [Online]. Available: https://www.microsoft.com/en-us/research/publication/introduction-neural-information-retrieval/.

[43] H. Li and J. Xu, "Semantic matching in search", *Found. Trends Inf. Retr.*, vol. 7, no. 5, pp. 343–469, Jun. 2014, ISSN: 1554-0669. DOI: 10.1561/1500000035. [Online]. Available: http://dx.doi.org/10.1561/1500000035.

[44] T.-Y. Liu, "Learning to rank for information retrieval", *Found. Trends Inf. Retr.*, vol. 3, no. 3, pp. 225–331, Mar. 2009, ISSN: 1554-0669. DOI: 10.1561/1500000016. [Online]. Available: http://dx.doi.org/10.1561/1500000016.

[45] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification", *CoRR*, vol. abs/1702.05659, 2017. arXiv: 1702.05659. [Online]. Available: http://arxiv.org/abs/1702.05659.

[46] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent", in *Proceedings of the 22Nd International Conference on Machine Learning*,

ser. ICML '05, Bonn, Germany: ACM, 2005, pp. 89–96, ISBN: 1-59593-180-5. DOI: 10.1145/1102351.1102363. [Online]. Available: http://doi.acm.org/10.1145/1102351.1102363.

[47] C. J. Burges, R. Ragno, and Q. V. Le, "Learning to rank with nonsmooth cost functions", in *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. C. Platt, and T. Hoffman, Eds., MIT Press, 2007, pp. 193–200. [Online]. Available: http://papers.nips.cc/paper/2971-learning-to-rank-with-nonsmooth-cost-functions.pdf.

[48] C. Burges, "From ranknet to lambdarank to lambdamart: An overview", *Learning*, vol. 11, Jan. 2010.

[49] Z. S. Harris, "Distributional Structure", *<i>WORD</i>*, vol. 10, no. 2-3, pp. 146–162, Aug. 1954, ISSN: 0043-7956. DOI: 10.1080/00437956.1954.11659520. [Online]. Available: http://www.tandfonline.com/doi/full/10.1080/00437956.1954.11659520.

[50] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space", Jan. 2013. arXiv: 1301.3781. [Online]. Available: https://arxiv.org/abs/1301.3781.

[51] B. Mitra, E. Nalisnick, N. Craswell, and R. Caruana, "A Dual Embedding Space Model for Document Ranking", Feb. 2016. arXiv: 1602.01137. [Online]. Available: http://arxiv.org/abs/1602.01137.

[52] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality", Oct. 2013. arXiv: 1310.4546. [Online]. Available: https://arxiv.org/abs/1310.4546.

[53] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model", in *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, R. G. Cowell and Z. Ghahramani, Eds., Society for Artificial Intelligence and Statistics, 2005, pp. 246–252. [Online]. Available: http://www.iro.umontreal.ca/~lisa/pointeurs/hierarchical-nnlm-aistats05.pdf.

[54] M. U. Gutmann and A. Hyvärinen, "Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics", *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 307–361, 2012, ISSN: ISSN 1533-7928. [Online]. Available: http://www.jmlr.org/papers/v13/gutmann12a.html.

[55] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information", *CoRR*, vol. abs/1607.04606, 2016. arXiv: 1607.04606. [Online]. Available: http://arxiv.org/abs/1607.04606.

[56] C. D. Manning, P. Raghavan, and H. Schütze, "Evaluation in information retrieval", in *Introduction to Information Retrieval*. Cambridge University Press, 2008, pp. 151–175. DOI: 10.1017/CBO9780511809071.007.

[57] A. Kosmopoulos, I. Androutsopoulos, and G. Paliouras, *Biomedical semantic indexing using dense word vectors in bioasq*, 2015. [Online]. Available: http://nlp.cs.aueb.gr/pubs/jbms_dense_vectors.pdf.

[58] G.-I. Brokos, P. Malakasiotis, and I. Androutsopoulos, "Using Centroids of Word Embeddings and Word Mover's Distance for Biomedical Document Retrieval in Question Answering", Aug. 2016. DOI: 10.18653/v1/W16-2915. arXiv: 1608.03905. [Online]. Available: https://arxiv.org/abs/1608.03905%20http://arxiv.org/abs/1608.03905.

[59] F. Galkó and C. Eickhoff, "Biomedical Question Answering via Weighted Neural Network Passage Retrieval", in, Springer, Cham, 2018, pp. 523–528. DOI: 10.1007/978-3-319-76941-7_39. [Online]. Available: http://link.springer.com/10.1007/978-3-319-76941-7_39.

[60] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning deep structured semantic models for web search using clickthrough data", in *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management - CIKM '13*, New York, New York, USA: ACM Press, 2013, pp. 2333–2338, ISBN: 9781450322638. DOI: 10.1145/2505515.2505665. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2505515.2505665.

[61] R. Salakhutdinov and G. Hinton, "Semantic hashing", *International Journal of Approximate Reasoning*, vol. 50, no. 7, pp. 969–978, 2009, Special Section on Graphical Models and Information Retrieval, ISSN: 0888-613X. DOI: https://doi.org/10.1016/j.ijar.2008.11.006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0888613X08001813.

[62] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval", in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management - CIKM '14*, New York, New York, USA: ACM Press, 2014, pp. 101–110, ISBN: 9781450325981. DOI: 10.1145/2661829.2661935. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2661829.2661935.

[63] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. Ward, "Deep Sentence Embedding Using Long Short-Term Memory Networks: Analysis and Application to Information Retrieval", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 4, pp. 694–707, Apr. 2016, ISSN: 2329-9290. DOI: 10.1109/TASLP.2016.2520371. [Online]. Available: http://ieeexplore.ieee.org/document/7389336/.

[64] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger, "From Word Embeddings to Document Distances", in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, JMLR.org, 2015, pp. 957–966. [Online]. Available: http://dl.acm.org/citation.cfm?id=3045118.3045221.

[65] Y. Rubner, C. Tomasi, and L. Guibas, "A metric for distributions with applications to image databases", in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, Narosa Publishing House, pp. 59–66, ISBN: 81-7319-221-9. DOI: 10.1109/ICCV.1998.710701. [Online]. Available: http://ieeexplore.ieee.org/document/710701/.

[66] S. Kim, W. J. Wilbur, and Z. Lu, "Bridging the gap: A semantic similarity measure between queries and documents", *CoRR*, vol. abs/1608.01972, 2016. arXiv: 1608.01972. [Online]. Available: http://arxiv.org/abs/1608.01972.

[67] J. Guo, Y. Fan, Q. Ai, and W. B. Croft, "A Deep Relevance Matching Model for Ad-hoc Retrieval", in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management - CIKM '16*, New York, New York, USA: ACM Press, 2016, pp. 55–64, ISBN: 9781450340731. DOI: 10.1145/2983323.2983769. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2983323.2983769.

[68] R. McDonald, G.-I. Brokos, and I. Androutsopoulos, "Deep Relevance Ranking Using Enhanced Document-Query Interactions", Sep. 2018. arXiv: 1809.01682. [Online]. Available: http://arxiv.org/abs/1809.01682.

[69] L. Pang, Y. Lan, J. Guo, J. Xu, J. Xu, and X. Cheng, "DeepRank", in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management - CIKM '17*, New York, New York, USA: ACM Press, 2017, pp. 257–266, ISBN: 9781450349185. DOI: 10.1145/3132847.3132914. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3132847.3132914.

[70] H. C. Wu, R. W. P. Luk, K. F. Wong, and K. L. Kwok, "A retrospective study of a hybrid document-context based retrieval model", *Inf. Process. Manage.*, vol. 43, no. 5, pp. 1308–1331, Sep. 2007, ISSN: 0306-4573. DOI: 10.1016/j.ipm.2006.10.009. [Online]. Available: http://dx.doi.org/10.1016/j.ipm.2006.10.009.

[71] S. Wan, Y. Lan, J. Xu, J. Guo, L. Pang, and X. Cheng, "Match-srnn: Modeling the recursive matching structure with spatial RNN", *CoRR*, vol. abs/1604.04378, 2016. arXiv: 1604.04378. [Online]. Available: http://arxiv.org/abs/1604.04378.

[72] M. Zhu, A. Ahuja, W. Wei, and C. K. Reddy, "A hierarchical attention retrieval model for healthcare question answering", in *The World Wide Web Conference*, ser. WWW '19, San Francisco, CA, USA: ACM, 2019, pp. 2472–2482, ISBN: 978-1-4503-6674-8. DOI: 10.1145/3308558.3313699. [Online]. Available: http://doi.acm.org/10.1145/3308558.3313699.

[73] M. J. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional attention flow for machine comprehension", *CoRR*, vol. abs/1611.01603, 2016. arXiv: 1611.01603. [Online]. Available: http://arxiv.org/abs/1611.01603.

[74] T. Qin and T. Liu, "Introducing LETOR 4.0 datasets", *CoRR*, vol. abs/1306.2597, 2013. arXiv: 1306.2597. [Online]. Available: http://arxiv.org/abs/1306.2597.

[75] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals,

P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *Tensorflow: Large-scale machine learning on heterogeneous distributed systems*, 2015. [Online]. Available: http://download.tensorflow.org/paper/whitepaper2015.pdf.

[76] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu, "Learning fine-grained image similarity with deep ranking", *CoRR*, vol. abs/1404.4661, 2014. arXiv: 1404.4661. [Online]. Available: http://arxiv.org/abs/1404.4661.

[77] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks", *CoRR*, vol. abs/1706.02515, 2017. arXiv: 1706.02515. [Online]. Available: http://arxiv.org/abs/1706.02515.

[78] Z. Yang, D. Yang, C. Dyer, X. He, A. J. Smola, and E. H. Hovy, "Hierarchical attention networks for document classification", in *HLT-NAACL*, 2016.

[79] *Bioasq evaluation measures for task b reference*, [Online; accessed 8. May. 2019], May 2019. [Online]. Available: participants-area.bioasq.org/Tasks/b/eval%5C_meas%5C_2018/.

[80] N. Cruz Diaz and M. M Maña López, "An analysis of biomedical tokenization: Problems and strategies", Sep. 2015. DOI: 10.18653/v1/W15-2605.

[81] Y. He and M. Kayaalp, *A comparison of 13 tokenizers on medline*, Dec. 2006. DOI: 10.13140/2.1.1133.1206.

[82] E. Charniak, "A maximum-entropy-inspired parser", in *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference*, ser. NAACL 2000, Seattle, Washington: Association for Computational Linguistics, 2000, pp. 132–139. [Online]. Available: http://dl.acm.org/citation.cfm?id=974305.974323.

[83] Y. Zhang, Q. Chen, Z. Yang, H. Lin, and Z. Lu, "BioWordVec, improving biomedical word embeddings with subword information and MeSH", *Sci. Data*, vol. 6, no. 1, p. 52, May 2019, ISSN: 2052-4463. DOI: 10.1038/s41597-019-0055-0.

[84] G.-I. Brokos, P. Liosis, R. McDonald, D. Pappas, and I. Androutsopoulos, "AUEB at BioASQ 6: Document and Snippet Retrieval", Sep. 2018. arXiv: 1809.06366. [Online]. Available: http://arxiv.org/abs/1809.06366.

APPENDIX A

Appendix A: Visualization of more extracted snippets

Tokenized query

hepadnaviral minichromosomes free nucleosomes

Tokenized document

characterization of nucleosome positioning in hepadnaviral covalently closed circular dna minichromosomes hepadnaviral covalently closed circular dna cccdna exists as an episomal minichromosome in the nucleus of virus infected hepatocytes and serves as the transcriptional template for the synthesis of viral mrnas to obtain insight on the structure of hepadnaviral cccdna minichromosomes we utilized ducks infected with the duck hepatitis b virus dhbv as a model and determined the in vivo nucleosome distribution pattern on viral cccdna by the micrococcal nuclease mnase mapping and genome wide pcr amplification of isolated mononucleosomal dhbv dna several nucleosome protected sites in a region of the dhbv genome nucleotides nt 2000 to 2700 known to harbor various cis transcription regulatory elements were consistently identified in all dhbv positive liver samples in addition we observed other nucleosome protection sites in dhbv minichromosomes that may vary among individual ducks but the pattern of mnase mapping in those regions is transmittable from the adult ducks to the newly infected ducklings these results imply that the nucleosomes along viral cccdna in the minichromosomes are not random but sequence specifically positioned furthermore we showed in ducklings that a significant portion of cccdna possesses a few negative superhelical turns suggesting the presence of intermediates of viral minichromosomes assembled in the liver where dynamic hepatocyte growth and cccdna formation occur this study supplies the initial framework for the understanding of the overall complete structure of hepadnaviral cccdna minichromosomes

Figure A.1: Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a positive document at position 2 of the resulting ranking list. Given the query *Are hepadnaviral minichromosomes free of nucleosomes?*

Tokenized query

hepadnaviral minichromosomes free nucleosomes

Tokenized document

polyoma virus minichromosomes a soluble in vitro replication system polyoma virus minichromosomes were isolated from infected 3t6 cells by hypotonic extraction of isolated nuclei the kinetics of in vitro dna synthesis in the nuclear extract was similar to that observed with intact nuclei the majority of the products of in vitro dna synthesis sedimented with replicative intermediate ri minichromosomes and migrated as two bands ri a and ri b on 1 4 agarose gels the kinetics of deoxynucleotide monophosphate incorporation into these species was consistent with the existence of several rate limiting steps in in vitro replication by polyoma minichromosomes electron microscope analysis showed that the ri a band consisted almost entirely of ri theta structures ranging from 46 to 87 replicated with one half of all theta structures 67 4 replicated the ri b material was more complex consisting of sigma and alpha structures with tails ranging from 7 to 114 of polyoma genome length and less frequently of linked and multiple linked dimeric structures

Figure A.2: Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a negative document at position 1000 of the resulting ranking list. Given the query *Are hepadnaviral minichromosomes free of nucleosomes?*

Tokenized query

kind analyses performed software tool unipept

Tokenized document

the unique peptidome taxon specific tryptic peptides as biomarkers for targeted metaproteomics the unique peptide finder http unipept ugent be peptidefinder is an interactive web application to quickly hunt for tryptic peptides that are unique to a particular species genus or any other taxon biodiversity within the target taxon is represented by a set of proteomes selected from a monthly updated list of complete and nonredundant uniprot proteomes supplemented with proprietary proteomes loaded into persistent local browser storage the software computes and visualizes pan and core peptidomes as unions and intersections of tryptic peptides occurring in the selected proteomes in addition it also computes and displays unique peptidomes as the set of all tryptic peptides that occur in all selected proteomes but not in any uniprot record not assigned to the target taxon as a result the unique peptides can serve as robust biomarkers for the target taxon for example in targeted metaproteomics studies computations are extremely fast since they are underpinned by the unipept database the lowest common ancestor algorithm implemented in unipept and modern web technologies that facilitate in browser data storage and parallel processing

Figure A.3: Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a positive document at position 2 of the resulting ranking list. Given the query *What kind of analyses are performed with the software tool "unipept"*

Tokenized query

kind analyses performed software tool unipept

Tokenized document

netlang a software for the linguistic analysis of corpora by means of complex networks to date there is no software that directly connects the linguistic analysis of a conversation to a network program networks programs are able to extract statistical information from data basis with information about systems of interacting elements language has also been conceived and studied as a complex system however most proposals do not analyze language according to linguistic theory but use instead computational systems that should save time at the price of leaving aside many crucial aspects for linguistic theory some approaches to network studies on language do apply precise linguistic analyses made by a linguist the problem until now has been the lack of interface between the analysis of a sentence and its integration into the network that could be managed by a linguist and that could save the analysis of any language previous works have used old software that was not created for these purposes and that often produced problems with some idiosyncrasies of the target language the desired interface should be able to deal with the syntactic peculiarities of a particular language the options of linguistic theory preferred by the user and the preservation of morpho syntactic information lexical categories and syntactic relations between items netlang is the first program able to do that recently a new kind of linguistic analysis has been developed which is able to extract a complexity pattern from the speakers linguistic production which is depicted as a network where words are inside nodes and these nodes connect each other by means of edges or links the information inside the edge can be syntactic semantic etc the netlang software has become the bridge between rough linguistic data and the network program netlang has integrated and improved the functions of programs used in the past namely the dga annotator and two scripts toxml pl and xml2pairs py used for transforming and pruning data netlang allows the researcher to make accurate linguistic analysis by means of syntactic dependency relations between words while tracking record of the nature of such syntactic relationships subject object etc the netlang software is presented as a new tool that solve many problems detected in the past the most important improvement is that netlang integrates three past applications into one program and is able to produce a series of file formats that can be read by a network program through the netlang software the linguistic network analysis based on syntactic analyses characterized for its low cost and the completely non invasive procedure aims to evolve into a sufficiently fine grained tool for clinical diagnosis in potential cases of language disorders

Figure A.4: Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a negative document at position 9 of the resulting ranking list. Given the query *What kind of analyses are performed with the software tool "unipept"*

Tokenized query

dnmt3 proteins present plants

Tokenized document

the de novo cytosine methyltransferase drm2 requires intact uba domains and a catalytically mutated paralog drm3 during rna directed dna methylation in arabidopsis thaliana eukaryotic dna cytosine methylation can be used to transcriptionally silence repetitive sequences including transposons and retroviruses this silencing is stable between cell generations as cytosine methylation is maintained epigenetically through dna replication the arabidopsis thaliana dnmt3 cytosine methyltransferase ortholog domains rearranged methyltransferase2 drm2 is required for establishment of small interfering rna sirna directed dna methylation in mammals piwi proteins and pirna act in a convergently evolved rna directed dna methylation system that is required to repress transposon expression in the germ line de novo methylation may also be independent of rna interference and small rnas as in neurospora crassa here we identify a clade of catalytically mutated drm2 paralogs in flowering plant genomes which in a thaliana we term domains rearranged methyltransferase3 drm3 despite being catalytically mutated drm3 is required for normal maintenance of non cg dna methylation establishment of rna directed dna methylation triggered by repeat sequences and accumulation of repeat associated small rnas although the mammalian catalytically inactive dnmt3l paralogs act in an analogous manner phylogenetic analysis indicates that the drm and dnmt3 protein families diverged independently in plants and animals we also show by site directed mutagenesis that both the drm2 n terminal uba domains and c terminal methyltransferase domain are required for normal rna directed dna methylation supporting an essential targeting function for the uba domains these results suggest that plant and mammalian rna directed dna methylation systems consist of a combination of ancestral and convergent features

Figure A.5: Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a positive document at position 1 of the resulting ranking list. Given the query *"Are there any DNMT3 proteins present in plants?"*

Tokenized query

dnmt3 proteins present plants

Tokenized document

the plant cyclin dependent kinase inhibitor ick1 has distinct functional domains for in vivo kinase inhibition protein instability and nuclear localization interactor inhibitor 1 of cdc2 kinase ick1 from arabidopsis thaliana is the first plant cyclin dependent kinase cdk inhibitor and overexpression of ick1 inhibits cdk activity cell division and plant growth in transgenic plants in this study ick1 and deletion mutants were expressed either alone or as green fluorescent protein gfp fusion proteins in transgenic arabidopsis plants deletion of the c terminal 15 or 29 amino acids greatly reduced or completely abolished the effects of ick1 on the transgenic plants and recombinant proteins lacking the c terminal residues lost the ability to bind to cdk complex and the kinase inhibition activity demonstrating the role of the conserved c terminal domain in in vivo kinase inhibition in contrast the mutant ick1deltan108 with the n terminal 108 residues deleted had much stronger effects on plants than the full length ick1 analyses demonstrated that this effect was not because of an enhanced ability of ick1deltan108 protein to inhibit cdk activity but a result of a much higher level of ick1deltan108 protein in the plants indicating that the n terminal domain contains a sequence or element increasing protein instability in vivo furthermore gfp ick1 protein was restricted to the nuclei in roots of transgenic plants even with the c terminal or the n terminal domain deleted suggesting that a sequence in the central domain of ick1 is responsible for nuclear localization these results provide mechanistic understanding about the function and regulation of this cell cycle regulator in plants

Figure A.6: Visualization of the query and snippets attention by the Self-Attn-DeepRank model for a positive document at position 1000 of the resulting ranking list. Given the query *"Are there any DNMT3 proteins present in plants?"*

APPENDIX B

Appendix B: Tensorflow graph visualization on TensorBoard

All the presented visualizations were automatically generated by the TensorBoard for the respective computational graph.

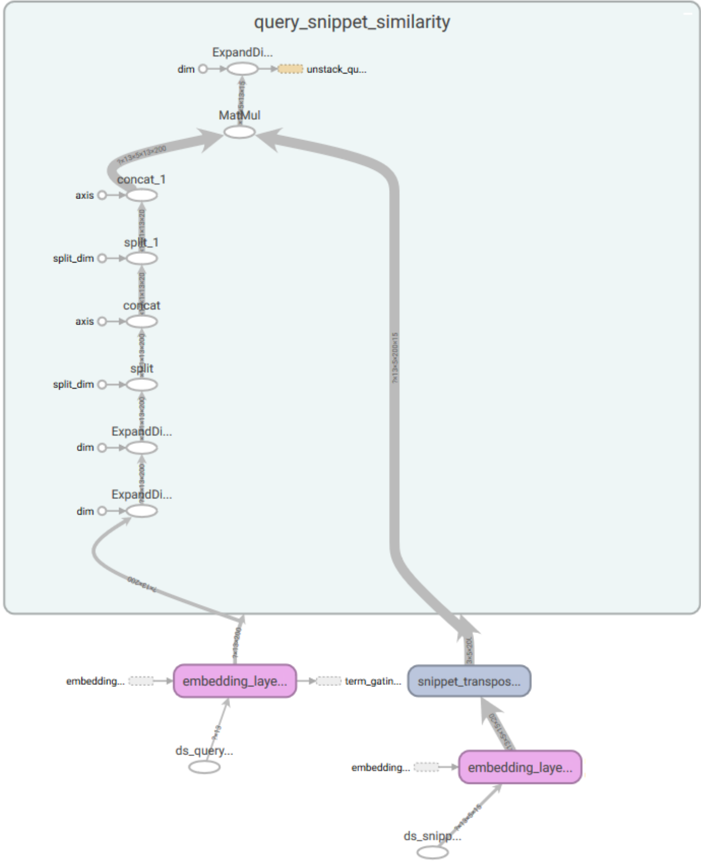


Figure B.1: Tensorboard visualization associated with implementation of Figure 5.6.

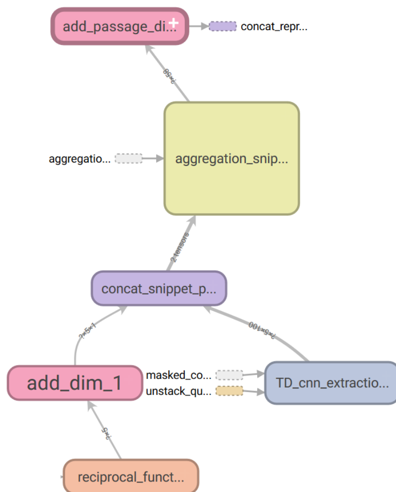


Figure B.2: Tensorboard visualization associated with implementation of Figure 5.7.

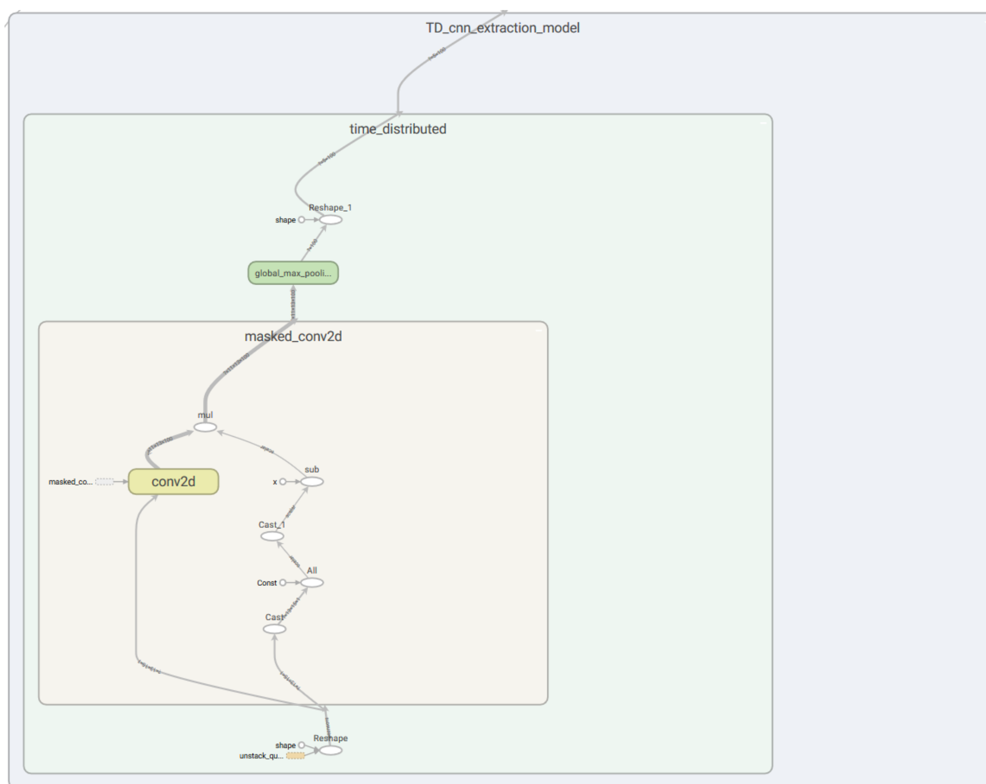
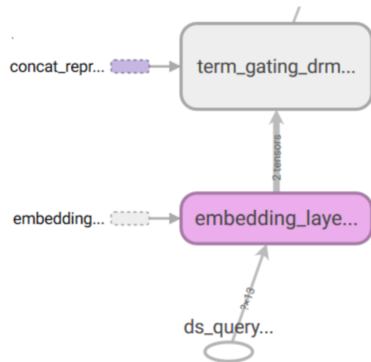
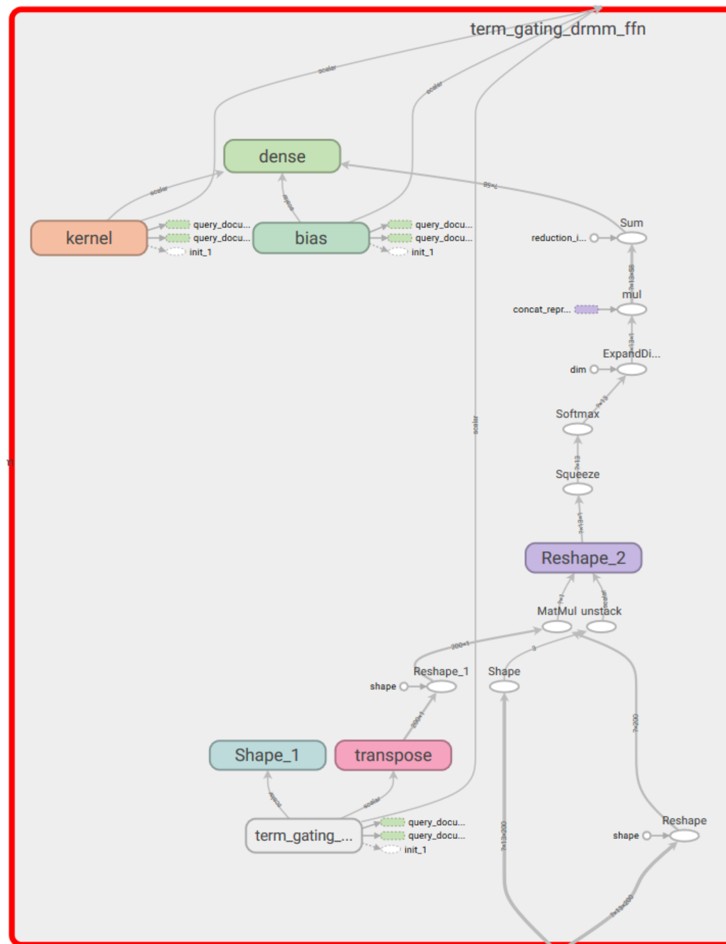


Figure B.3: Tensorboard visualization of the CNN extraction sub-model



a) Aggregation network



b) Expansion of the term gatting graph

Figure B.4: Tensorboard visualization associated with implementation of Figure 5.8.

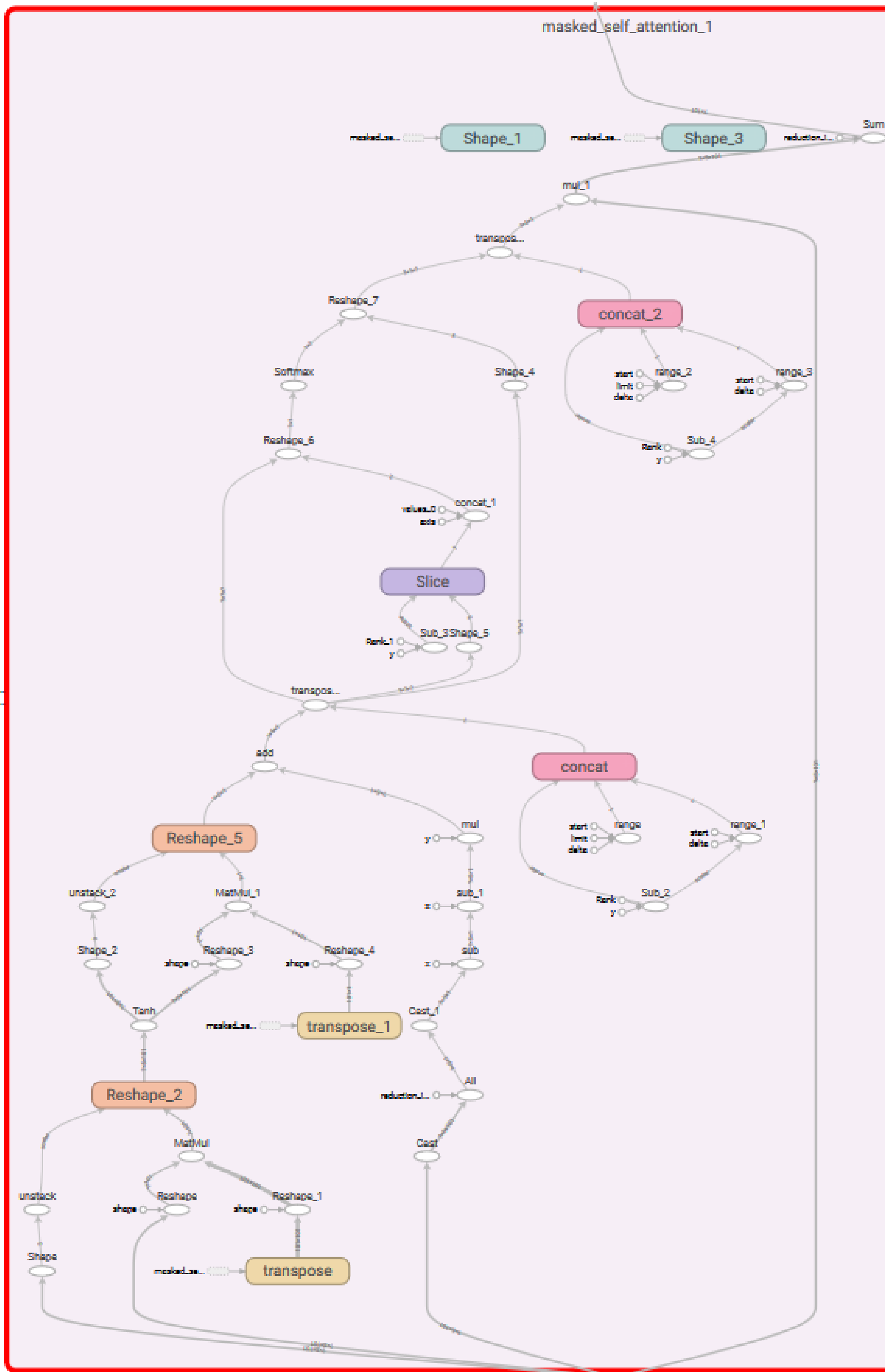
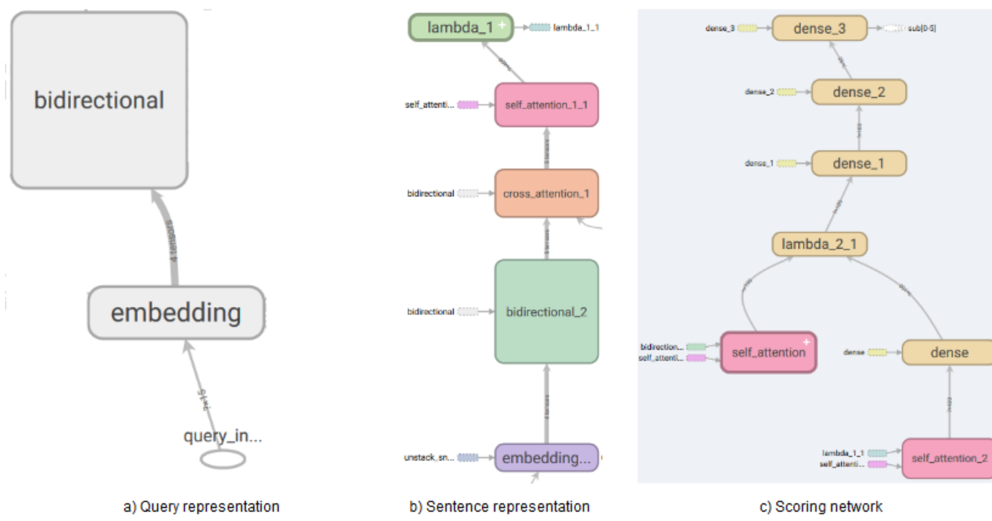


Figure B.5: Tensorboard visualization associated with implementation of Figure 5.12.



a) Query representation b) Sentence representation c) Scoring network

Figure B.6: Tensorboard visualization associated with implementation of Figure 5.13. The output of a) and b) are fed to the self_attention and self_attention_2 layers presented in c).

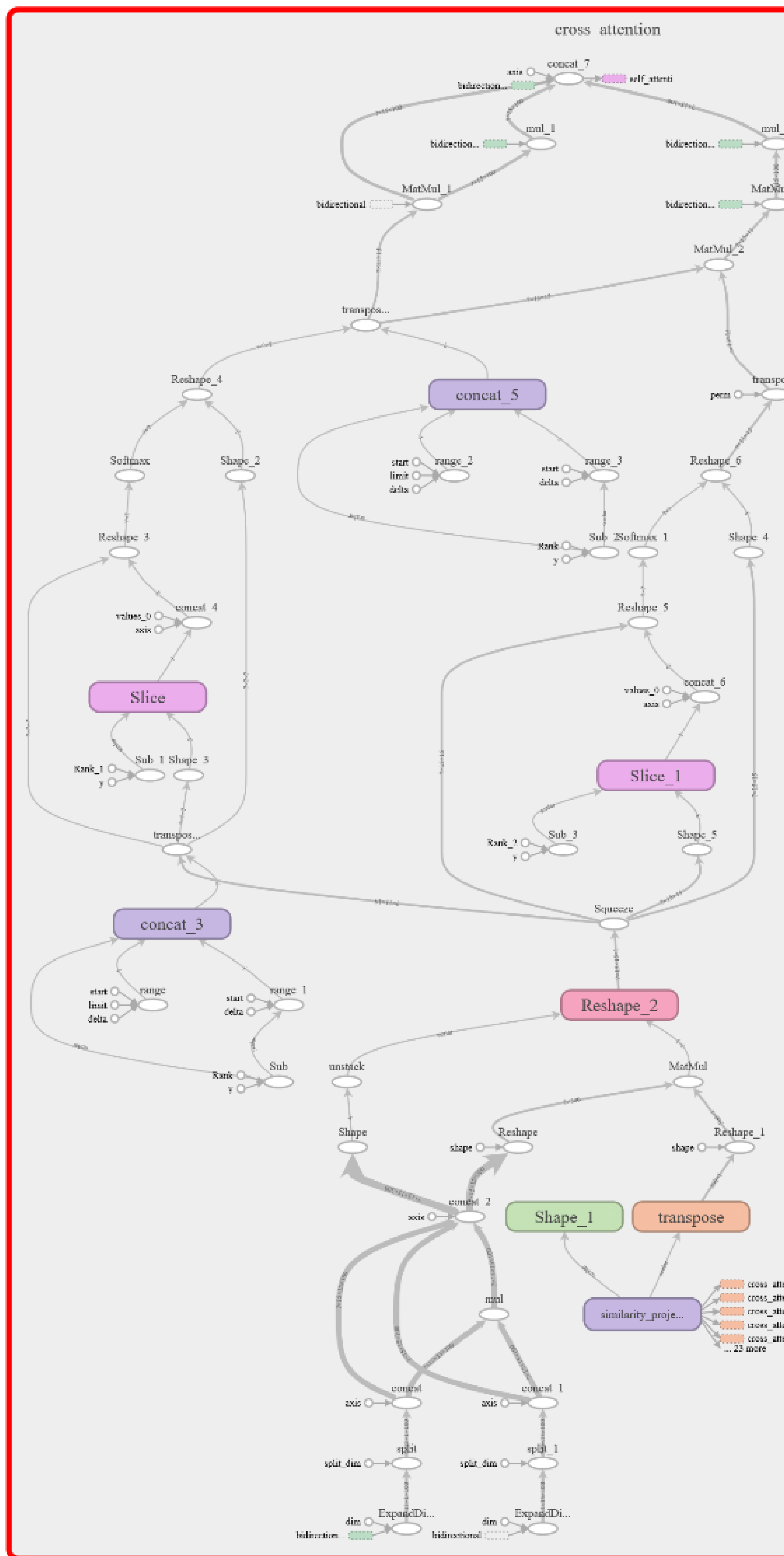


Figure B.7: Tensorboard visualization associated with implementation of Figure 5.14

APPENDIX C

Appendix C: Implementation of the custom layers in *Keras*

```
class SimilarityMatrix(Layer):
    def __init__(self, query_max_term, snippet_max_term, interaction_mode=0, **kwargs):
        """
        interaction mode 0: only use similarity matrix
                    mode 1: similarity matrix + query and snippet embeddings
        """
        assert interaction_mode in [0,1] #only valid modes

        self.query_max_term = query_max_term
        self.snippet_max_term = snippet_max_term
        self.interaction_mode = interaction_mode

        super().__init__(**kwargs)

    def call(self,x):
        if self.interaction_mode==0:
            #sim => dot product (None, MAX_Q_TERM, EMB_DIM) x (None, MAX_Q_TERM, MAX_PASSAGE_PER_Q, EMB_DIM,
            query = K.expand_dims(x[0], axis=1) #(None, 1, MAX_Q_TERM, EMB_DIM)
            query = K.expand_dims(query, axis=1) #(None, 1, 1, MAX_Q_TERM, EMB_DIM)
            query = K.repeat_elements(query,x[1].shape[1],axis=1) #(None, MAX_PASSAGE_PER_Q, MAX_Q_TERM, EMB_
            query = K.repeat_elements(query,x[1].shape[2],axis=2)
            s_matrix = K.batch_dot(query,x[1]) #(None, MAX_PASSAGE_PER_Q, MAX_Q_TERM, #(None, MAX_PASSAGE_PE

            s_matrix = K.expand_dims(s_matrix)

            return s_matrix #Add one more dimension #(None, MAX_PASSAGE_PER_Q, MAX_Q_TERM, #(None, MAX_PASSA
        elif self.interaction_mode==1:
            raise NotImplementedError("interaction mode of layer SimilarityMatrix is not implemented")
```

Code 1: Implementation of the similarity layer.

```
class MaskedConv2D(Layer):

    def __init__(self, filters, kernel_size, activation, regularizer=None, **kargs):
        super(MaskedConv2D, self).__init__(**kargs)

        self.activation = activations.get(activation)

        if regularizer is None or isinstance(regularizer,str):
            self.regularizer = regularizers.get(regularizer)
        else:
            self.regularizer = regularizer

        self.filters = filters
        self.kernel_size = kernel_size
        self.activation = activation

    def build(self, input_shape):
        #NOTE THAT:
        #This class does not extend Conv2D
        #because it is more simple to add more Conv2D in parallel
        #this way, (future updates)
        self.conv2dlayer = Conv2D( filters = self.filters, kernel_size=self.kernel_size, activation=self.activat
        self.conv2dlayer.build(input_shape)
        self._trainable_weights = self.conv2dlayer.trainable_weights

        super(MaskedConv2D, self).build(input_shape)

    def call(self, x):

        condition = K.all(x) #if all the values are the same
        inv_condition = (1-K.cast(condition, K.floatx()))
        print(inv_condition)
        feature_maps = self.conv2dlayer(x)

        return feature_maps * inv_condition
```

Code 2: Implementation of the masked conv2d layer.

```
class TermGatingDRMM(Layer):

    def __init__(self, embedding_dim, rnn_dim, activation=None, initializer='glorot_normal', regularizer=None
        super(TermGatingDRMM_FFN, self).__init__()

        self.activation = activations.get(activation)
        self.initializer = initializers.get(initializer)

        if regularizer is None or isinstance(regularizer,str):
            self.regularizer = regularizers.get(regularizer)
        else:
            self.regularizer = regularizer

        self.emb_dim = embedding_dim
        self.rnn_dim = rnn_dim

    def build(self, input_shape):

        #term gating W
        self.W_query = self.add_variable(name = "term_gating_We",
                                shape = [self.emb_dim,1],
                                initializer = self.initializer,
                                regularizer = self.regularizer,)

        super(TermGatingDRMM_FFN, self).build(input_shape)

    def call(self, x):

        query_embeddings = x[0] #(None, MAX_Q_TERM, EMB_SIZE)
        snippet_representation_per_query = x[1] #(None, MAX_Q_TERM, BI_GRU_DIM)

        #compute gated weights
        gated_logits = K.squeeze(K.dot(query_embeddings, self.W_query), axis = -1 )
        #print(gated_logits)
        gated_distribution = K.expand_dims(K.softmax(gated_logits))
        #print(gated_distribution)
        #snippet projection
        self.attention_weights = gated_distribution

        weighted_rep = K.sum(snippet_representation_per_query * gated_distribution,  axis = 1)
        print(weighted_rep)

        return weighted_rep
```

Code 3: Implementation of the term gating layer.

```
class SelfAttention(Layer):

    def __init__(self, attention_dimension, initializer='glorot_normal', regularizer=None, **kargs):
        super(SelfAttention, self).__init__(**kargs)
        self.initializer = initializer
        self.attention_dimension = attention_dimension
        if regularizer is None or isinstance(regularizer,str):
            self.regularizer = regularizers.get(regularizer)
        else:
            self.regularizer = regularizer

    def build(self, input_shape):
        emb_dim = int(input_shape[2])
        self.W_attn_project = self.add_variable(name = "self_attention_projection",
                                   shape = [emb_dim, 1],
                                   initializer = self.initializer,
                                   regularizer = self.regularizer,)
        self.W_attn_score = self.add_variable(name = "self_attention_score",
                                   shape = [self.attention_dimension, 1],
                                   initializer = self.initializer,
                                   regularizer = self.regularizer,)
        super(SelfAttention, self).build(input_shape)

    def call(self, x):
        x_projection = K.dot(x, self.W_attn_project) # (NONE, 300, 300)
        x_tanh = K.tanh(x_projection) # (NONE, 300, 15)
        x_attention = K.dot(x_tanh, self.W_attn_score)
        x_attention_softmax = K.softmax(x_attention,axis = 1)
        x_scored_emb = x_attention_softmax * x
        return K.sum(x_scored_emb, axis=1)
```

Code 4: Implementation of the self-attention layer.

```python
class CrossAttention(Layer):
    def __init__(self, initializer='glorot_normal', regularizer=None, **kargs):
        super(CrossAttention, self).__init__(**kargs)
        self.initializer = initializer
        if regularizer is None or isinstance(regularizer,str):
            self.regularizer = regularizers.get(regularizer)
        else:
            self.regularizer = regularizer

    def build(self, input_shape):
        """
        input: [0] - query context embedding
               [1] - document context embedding
        """
        doc_embedding = input_shape[1]
        query_embedding = input_shape[0]
        self.query_len = query_embedding[1]
        self.doc_len = doc_embedding[1]
        assert int(query_embedding[2]) == int(doc_embedding[2])
        self.embedding_dim = int(query_embedding[2])
        self.W_sim_projection = self.add_variable(name = "similarity_projection",
                                    shape = [self.embedding_dim*3,1],
                                    initializer = self.initializer,
                                    regularizer = self.regularizer,)
        super(CrossAttention, self).build(input_shape)

    def call(self, x):
        """
        input: [0] - query context embedding
               [1] - document context embedding
        """
        doc_embedding = x[1]
        query_embedding = x[0]
        #build similarity matrix | row document token | colum query token
        doc_q_matrix = K.expand_dims(doc_embedding, axis=2)
        doc_q_matrix = K.repeat_elements(doc_q_matrix, self.query_len, axis=2)
        q_doc_matrix = K.expand_dims(query_embedding, axis=1)
        q_doc_matrix = K.repeat_elements(q_doc_matrix, self.doc_len, axis=1)

        element_mult = doc_q_matrix * q_doc_matrix
        #concatenation
        S = K.concatenate([doc_q_matrix, q_doc_matrix, element_mult])
        S = K.dot(S, self.W_sim_projection)
        S = K.squeeze(S,axis=-1)
        S_D2Q = K.softmax(S, axis=1)
        S_Q2D = K.softmax(S, axis=2)
        A_D2Q = K.batch_dot(S_D2Q, query_embedding)
        S_Q2D_transpose = K.permute_dimensions(S_Q2D,[0,2,1])
        A_D2Q_Q2D = K.batch_dot(S_D2Q, S_Q2D_transpose)
        A_Q2D = K.batch_dot(A_D2Q_Q2D, doc_embedding)
        #concat
        doc_attn = doc_embedding * A_D2Q
        doc_q_attn = doc_embedding * A_Q2D

        return K.concatenate([doc_embedding, A_D2Q, doc_attn, doc_q_attn])
```

Code 5: Implementation of the cross attention layer.

```
class ModelAPI():
    """
    API to interact in a unified way with all neural an non neural models
    """
    def __init__(self, saved_models_path='/backup/saved_models/' , metrics=[f_map,f_recall]):
        self.trained = False
        self.metrics = metrics
        self.saved_models_path = saved_models_path

    def train(self, data,  **kargs):
        self._training_process(data, **kargs)
        self.trained = True

    def _training_process(self, data, **kargs):
        raise NotImplementedError

    def predict(self, data, **kargs):
        if not self.trained:
            raise RuntimeError("The models most be trained before the inference")
        return self._predict_process(data, **kargs)

    def _predict_process(self, data, **kargs):
        raise NotImplementedError

    def evaluate(self, queries, expectations, index_pmid_mapping = None, batch_size=None):
        """
        run set of metrics over a data
        """
        (... hidden ...)

        return [ (metric.__name__,metric(predictions,expectations)) for metric in self.metrics]

    def save(self, **kargs):
        raise NotImplementedError

    @staticmethod
    def load(**kargs):
        raise NotImplementedError
```

Code 6: Interface that all the neural an non neural models should implement.

APPENDIX D

Appendix D: Complete results of the BioASQ TaskB phase A

Tables with the current results of the BioASQ TaskB phase. However a online document was also created in order to reflect future updates[a] .

[a]https://docs.google.com/spreadsheets/d/1M6MPP5PyRh9jGwhSuqQRYdfvGt8lpvhD3XxJJ1FJbgI/edit?usp=sharing

Table D.1: Table with the results of the first test batch for the BioASQ TaskB phase A, the developed systems are highlighted in bold.

System	Mean precision	Recall	F-Measure	MAP	GMAP
System v1	**0.1641**	**0.5544**	**0.2205**	**0.0874**	**0.0052**
System v2	**0.1621**	**0.5558**	**0.2181**	**0.0865**	**0.0063**
Ir_sys2	0.1190	0.5216	0.1746	0.0809	0.0047
lh_sys4	0.1200	0.5069	0.1745	0.0798	0.0039
lh_sys1	0.1200	0.5069	0.1745	0.0795	0.0041
lh_sys5	0.1220	0.5192	0.1778	0.0792	0.0046
Deep ML methods for	0.1120	0.5087	0.1660	0.0742	0.0039
lh_sys2	0.1060	0.4348	0.1531	0.0721	0.0021
Ir_sys1	0.1110	0.4887	0.1637	0.0676	0.0033
lh_sys3	0.1050	0.4608	0.1541	0.0616	0.0024
auth-qa-1	0.1920	0.2770	0.2069	0.0611	0.0003
Ir_sys3	0.0870	0.3620	0.1256	0.0444	0.0010
Ir_sys4	0.0900	0.3762	0.1306	0.0415	0.0012
lalala	0.1406	0.2264	0.1571	0.0386	0.0002

Table D.2: Table with the results of the second test batch for the BioASQ TaskB phase A, the developed systems are highlighted in bold.

System	Mean precision	Recall	F-Measure	MAP	GMAP
aueb-nlp-4	0.1270	0.5908	0.1911	0.0849	0.0067
aueb-nlp-5	0.2872	0.5663	0.3443	0.0830	0.0049
lalala	0.1250	0.5779	0.1877	0.0791	0.0048
lh_sys4	0.1230	0.5702	0.1846	0.0772	0.0045
aueb-nlp-3	0.1260	0.5967	0.1905	0.0771	0.0075
lh_sys1	0.1250	0.5779	0.1877	0.0768	0.0046
System v2	**0.1200**	**0.5619**	**0.1815**	**0.0764**	**0.0047**
System v1	**0.1210**	**0.5691**	**0.1829**	**0.0760**	**0.0051**
lh_sys3	0.1250	0.5779	0.1877	0.0753	0.0044
Ir_sys4	0.1333	0.5679	0.1964	0.0752	0.0045
aueb-nlp-1	0.1170	0.5776	0.1784	0.0741	0.0066
aueb-nlp-2	0.1200	0.5777	0.1823	0.0741	0.0062
lh_sys5	0.1150	0.5403	0.1737	0.0705	0.0040
lh_sys2	0.1120	0.5479	0.1716	0.0692	0.0041
Ir_sys1	0.1030	0.4687	0.1568	0.0662	0.0026
Ir_sys2	0.1250	0.5779	0.1877	0.0619	0.0036
Ir_sys3	0.1250	0.5779	0.1877	0.0601	0.0035
Deep ML methods for	0.0950	0.4733	0.1444	0.0579	0.0021
auth-qa-1	0.1760	0.2990	0.2020	0.0569	0.0004
from milab	0.0420	0.1883	0.0631	0.0267	0.0001
MindLab Red Lions++	0.0140	0.0825	0.0214	0.0105	0.0000
MindLab QA Reloaded	0.0140	0.0825	0.0214	0.0105	0.0000
MindLab QA System ++	0.0140	0.0825	0.0214	0.0104	0.0000
MindLab QA System	0.0140	0.0825	0.0214	0.0104	0.0000

Table D.3: Table with the results of the third test batch for the BioASQ TaskB phase A, the developed systems are highlighted in bold.

System	Mean precision	Recall	F-Measure	MAP	GMAP
aueb-nlp-4	0.1270	0.5908	0.1911	0.0849	0.0067
aueb-nlp-5	0.2872	0.5663	0.3443	0.0830	0.0049
lalala	0.1250	0.5779	0.1877	0.0791	0.0048
lh_sys4	0.1230	0.5702	0.1846	0.0772	0.0045
aueb-nlp-3	0.1260	0.5967	0.1905	0.0771	0.0075
lh_sys1	0.1250	0.5779	0.1877	0.0768	0.0046
System v2	**0.1200**	**0.5619**	**0.1815**	**0.0764**	**0.0047**
System v1	**0.1210**	**0.5691**	**0.1829**	**0.0760**	**0.0051**
lh_sys3	0.1250	0.5779	0.1877	0.0753	0.0044
Ir_sys4	0.1333	0.5679	0.1964	0.0752	0.0045
aueb-nlp-1	0.1170	0.5776	0.1784	0.0741	0.0066
aueb-nlp-2	0.1200	0.5777	0.1823	0.0741	0.0062
lh_sys5	0.1150	0.5403	0.1737	0.0705	0.0040
lh_sys2	0.1120	0.5479	0.1716	0.0692	0.0041
Ir_sys1	0.1030	0.4687	0.1568	0.0662	0.0026
Ir_sys2	0.1250	0.5779	0.1877	0.0619	0.0036
Ir_sys3	0.1250	0.5779	0.1877	0.0601	0.0035
Deep ML methods for	0.0950	0.4733	0.1444	0.0579	0.0021
auth-qa-1	0.1760	0.2990	0.2020	0.0569	0.0004
from milab	0.0420	0.1883	0.0631	0.0267	0.0001
MindLab Red Lions++	0.0140	0.0825	0.0214	0.0105	0.0000
MindLab QA Reloaded	0.0140	0.0825	0.0214	0.0105	0.0000
MindLab QA System ++	0.0140	0.0825	0.0214	0.0104	0.0000
MindLab QA System	0.0140	0.0825	0.0214	0.0104	0.0000

Table D.4: Table with the results for the fourth test batch of the BioASQ TaskB phase A, the developed systems are highlighted in bold.

System	**Mean precision**	**Recall**	**F-Measure**	**MAP**	**GMAP**
aueb-nlp-4	0.1461	0.6132	0.2148	0.1034	0.0112
aueb-nlp-5	0.3332	0.6141	0.3783	0.1015	0.0116
aueb-nlp-2	0.1391	0.5995	0.2051	0.0968	0.0083
aueb-nlp-1	0.1391	0.6139	0.2056	0.0951	0.0101
System v1	**0.1331**	**0.5719**	**0.1934**	**0.0922**	**0.0054**
aueb-nlp-3	0.1311	0.5879	0.1942	0.0909	0.0083
System v2	**0.1291**	**0.5667**	**0.1887**	**0.0882**	**0.0054**
lh_sys4	0.1240	0.5599	0.1853	0.0835	0.0051
lh_sys2	0.1200	0.5510	0.1796	0.0812	0.0044
lh_sys3	0.1130	0.5228	0.1695	0.0793	0.0032
lh_sys5	0.1160	0.5342	0.1737	0.0771	0.0039
lh_sys1	0.1140	0.5297	0.1709	0.0768	0.0035
MindLab QA Reloaded	0.1040	0.5103	0.1573	0.0726	0.0033
MindLab QA System ++	0.1040	0.5103	0.1573	0.0726	0.0033
MindLab QA System	0.1040	0.5103	0.1573	0.0726	0.0033
MindLab Red Lions++	0.1040	0.5103	0.1573	0.0726	0.0033
Deep ML methods for	0.0980	0.4795	0.1481	0.0710	0.0028
auth-qa-1	0.1790	0.3316	0.2094	0.0605	0.0006

Table D.5: Table with the results for the fifth test batch of the BioASQ TaskB phase A, the developed systems are highlighted in bold.

System	Mean precision	Recall	F-Measure	MAP	GMAP
aueb-nlp-4	0.0710	0.3937	0.1120	0.0425	0.0010
aueb-nlp-5	0.1751	0.3670	0.2012	0.0399	0.0008
System v2	**0.0640**	**0.3746**	**0.1016**	**0.0373**	**0.0007**
aueb-nlp-1	0.0620	0.3614	0.0990	0.0368	0.0006
lh_sys3	0.0570	0.3700	0.0927	0.0368	0.0005
aueb-nlp-3	0.0620	0.3667	0.0991	0.0366	0.0007
aueb-nlp-2	0.0560	0.3395	0.0900	0.0355	0.0006
lh_sys5	0.0590	0.3794	0.0962	0.0354	0.0006
lh_sys1	0.0570	0.3663	0.0931	0.0347	0.0005
System v1	**0.0620**	**0.3843**	**0.0995**	**0.0344**	**0.0008**
lh_sys4	0.0590	0.3741	0.0963	0.0331	0.0006
Deep ML methods for	0.0530	0.3420	0.0859	0.0329	0.0005
lh_sys2	0.0570	0.3404	0.0920	0.0328	0.0005
MindLab QA System ++	0.0550	0.3476	0.0888	0.0326	0.0005
MindLab Red Lions++	0.0550	0.3476	0.0888	0.0326	0.0005
MindLab QA System	0.0550	0.3476	0.0888	0.0326	0.0005
MindLab QA Reloaded	0.0550	0.3476	0.0888	0.0326	0.0005
auth-qa-1	0.0910	0.2067	0.1167	0.0223	0.0001
auth-qa-2	0.0070	0.0414	0.0112	0.0032	0.0000