UNIVERSITÉ DE LORRAINE

BERT Model and Convolutional Neural Networks for Relation Extraction

Author:
Fatima HABIB

Supervisors:
Prof. Yannick TOUSSAINT
PhD. Student Laura ZANELLA

September 8, 2021

A thesis submitted in fulfillment of the requirements for master 2 degree of Natural Language Processing

at

Institut des sciences du digital Management Cognition (IDMC)

Declaration of Authorship

I, Fatima HABIB, declare that this thesis titled, "BERT Model and Convolutional Neural Networks for Relation Extraction" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Fatima HABIB

Date:August 26, 2021

"I would like to thank my supervisors, Mr. Yannick Toussaint and Mrs.Laura Zanella. Their expertises were invaluable in formulating my scientific questions and answering them. The insightful feedback of Mr.Toussaint helped a lot during this internship to sharpen my thinking and further improving my scientific knowledge in the natural language processing domain. So, I deeply thank him for his great supervision.

I would like also to warmly thank Mr.Miguel Couceiro and Mr.Maxime Amblard for their guidance and support during our master program.

Finally, I would also like to thank my family and my friends for their support.
"

Fatima HABIB

UNIVERSITÉ DE LORRAINE

Abstract

Institut des sciences du digital Management Cognition (IDMC)

Natural Language Processing

BERT Model and Convolutional Neural Networks for Relation Extraction

by Fatima HABIB

The volume of biomedical literature is growing rapidly. It contains large amount of information about proteins/genes, drugs/chemicals, and their interactions. This fast growth creates the need to have a system able to retrieve these interactions relevant for precision medicine, and can help researchers in the biomedical domain.

In this master internship, we focus on building a relations extraction system to automatically extract relations between proteins / genes and drugs / chemicals (P/G-D/C interactions). To do so, we studied relations extraction task applied to biomedical PubMed abstracts (DrugProt corpus).

We suggested three model architectures: convolutional neural networks (CNNs) based models, Bidirectional Encoder Representations from Transformers (BERT) based models, and hybrid models (BERT + CNN). Our results show that the hybrid models achieved higher performance compared to the other two models, which can partially attributed to the rich contextualized content of the language model provided by BERT, introduced and detailed in chapter 4. The hybrid-based models outperform those of CNNs by approximately 10% in accuracy. The more accurate results obtained by hybrid models have been also reached in a smaller number of epochs compared to those in CNN based models, which can be attributed to high contextualized representations and parallelization degree of this model.

Contents

List of Abbreviations

NLP	**N**atural **L**anuage **P**rocessing
DL	**D**eep **L**earning
RE	**R**elation **E**xtraction
TD-IDF	**T**erm **F**requency-**I**nverse **D**ocument **F**requency
DS-R	**D**istributional **S**imilarity based **R**epresentations
word2vec	**word to vec**tor embedding
GloVe	**Glo**bal **Ve**ctors for word representation
Elmo	**E**mbeddings from **L**anguage **Mo**del
NN	**N**eural **N**etworks
FC	**F**ully **C**onnected **L**ayer
CNN	**C**onvolutional **N**eural **N**etworks
RNN	**R**ecurrent **N**eural **N**etworks
LSTM	**L**ong **S**hort **T**erm **M**emory
Seq2Seq	**S**equence **to S**equence model
BERT	**B**idirectional **E**ncoder **R**epresentations from **T**ransformers
LM	**L**anguage **M**odel
MLM	**M**asked **L**anguage **M**odel
NSP	**N**ext **S**entence **P**rediction

Chapter 1

Introduction to Convolutional Neural Networks

In this chapter, we are going to briefly review the basics of neural networks (NN) and convolutional neural networks (CNN). We start by introducing the neural network concept, and then we present the idea of the convolutional neural network using examples from both image recognition and natural language processing fields. The spirit of this chapter is greatly inspired by the book of Nielsen "Neural Networks and Deep Learning" [1].

1.1 Introduction

Deep learning (DL) is a branch of machine learning (ML) which employs neural networks (NN) techniques to process data and perform various tasks, such as image recognition, speech recognition, and NLP.

Neural networks (NN) are very strong, biologically-inspired programming tools which allows the machine to learn from observations, where the efficiency of this learning process is measured by a "*cost function*" that must be minimized as to improve the NN performance (*learning process*). The learning process can be "*supervised*" or "*unsupervised*". In supervised learning, the algorithms, used by the NN, are trained using labeled datasets which allows improving the accuracy of the NN when used to classify data or predict the outcomes. On the other hand, in the unsupervised learning, the algorithms learn patterns from un-labeled data without any human intervention [1]–[3] .

In general, the main components of a neural network are

1- Architecture: any neural network consists of different layers where each layer passes the processed data to the next layer. The first layer is known as the *input layer* which receives the input data used to make a prediction or to train the network. The last layer is called the *output layer* giving the outcomes of NN. Layers exist between the input and output layers are called the hidden layers. Each layer is composed of a number of computational units known as *neurons*. Each neuron is characterized by two parameters: the weight and bias. The main objective of the learning process is to automatically tune the weights and the biases of the NN neurons so to maximize the accuracy of its outcomes.

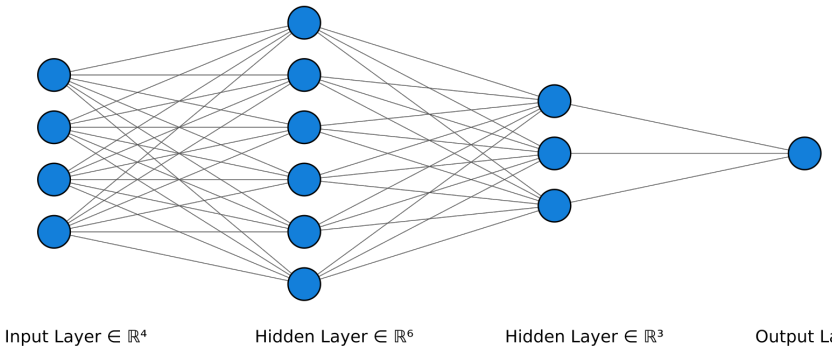

FIGURE 1.1: A visualization of a feed forward neural network. The NN consists of four layers: an input layer with four neurons, first hidden layer with 6 neurons, second hidden layer with 3 neurons, and an one-neuron output layer. This figure was plot on NN-SVG website [4].

Fig.1.1 shows a feed forward NN (or *fully-connected NN*) composed of four layers: input layer, 2 hidden layers, and an output layer.

2- Activation functions: after receiving data from the previous layer, each neuron process the data using an activation function. There are different types of activation functions: for example, the linear perceptrons can be regarded as threshold activation functions, meaning that they can be either "ON" or "OFF". Another example is the nonlinear activation functions that allow considering more complicated problems, reduce the complexity of the required NN, and account for any nonlinearity inherited in the studied problem.

3- Cost function: it is a non-negative function measuring the accuracy of the outcomes of the NN. Depending on the problem nature, there are also different type of cost function, such as mean square cost function (*MSE*), or the *cross-entropy* cost function.

4- Learning algorithm (*optimization algorithm*): It represents the learning mechanism by which the NN adjusts its weights and biases so that to minimize the cost function. There are two main optimization algorithms: gradient-based optimizers, such as the stochastic gradient decent (SGD), and derivative-free optimizers.

1.2 Convolutional Neural Networks

In the fully-connected NN, the neurons of each layer have the same design: each neuron is connected to all other neurons in the adjacent layers, as shown in Fig.1.1, and equipped with the same activation function. However, neurons in different layers may have different activation functions depending on the task, or the feature, that layer attempts to learn.

The previous discussion implies that fully-connected NNs are dense, and therefore, computationally expensive. For example, for the NN shown in

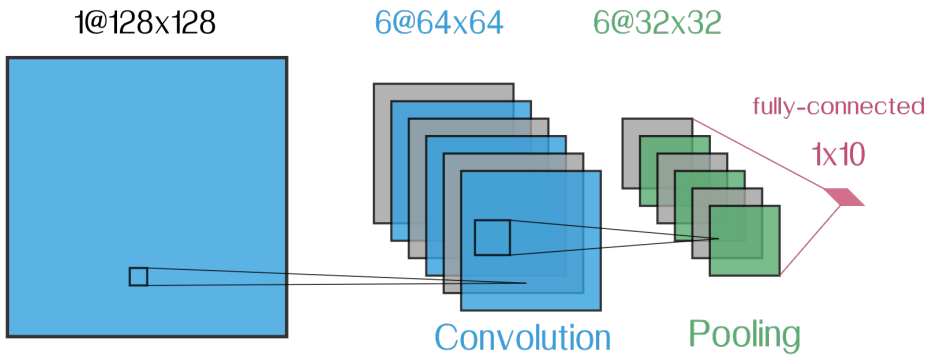

FIGURE 1.2: An illustration of a convolutional neural network consisting of an input layer of size 128×128 neurons, convolutional hidden layer composed 64×64 with the capability of detecting 6 features, a pooling layer for detected features, and an output fully-connected layer composed of 10 neurons. This figure was plot on NN-SVG website [4].

Fig.1.1, there are 55 unknown variables [1] to be determined using the optimization algorithm. However, this number grows rapidly for deeper NNs used for studying real-life problems, such as those in image recognition or NLP. For example, for a fully-connected NN composed of four subsequent layers to recognize handwritten digits can be composed of 784 (the input vector of the digit image which is originally a 28×28 pixels in MNIST dataset [5]), 64 (the number of neurons in the first hidden layer) , 64 (the number of neurons in the second hidden layer), 10 (the number of neurons in the output layer, where the output of each neuron here gives the probability of having an image with the corresponding number. For example, the 3^d gives the probability of having an image of 3.) neurons, respectively. The number of unknown variables becomes to 55834 - a huge increase!

Another property characterize the fully connected NNs: the lack of data structure recognition. Since the fully-connected NN makes no difference between neurons, then it does not pay attention to the spatial structure (or in general, the structure) of the data. For example, if the input are images (or sentences), then, in the input layer, the fully-connected NN treats the input pixels which are far apart and close together on exactly the same manner (or words in a sentence).

Therefore, a NN with an architecture that takes into account the data structure and reduce the number of unknown variables will improve the performance of the learning process. Here is comes the convolutional neural networks CNNs. The concept of CNN goes back to the 1970s [6]. At that time, Fukushima designed neural network with multiple pooling and convolutional layers (*Neocognitron*) to recognize visual patterns [7]. However,

[1] In Fig.1.1, there are 4, 6, 3, and 1 neurons in the four subsequent layers of the NN. This means that the number of biases is 10. While we have 3 weight matrices with 24, 18 and 3 weights, respectively. This in turn implies that the total numbers of unknowns is 55.

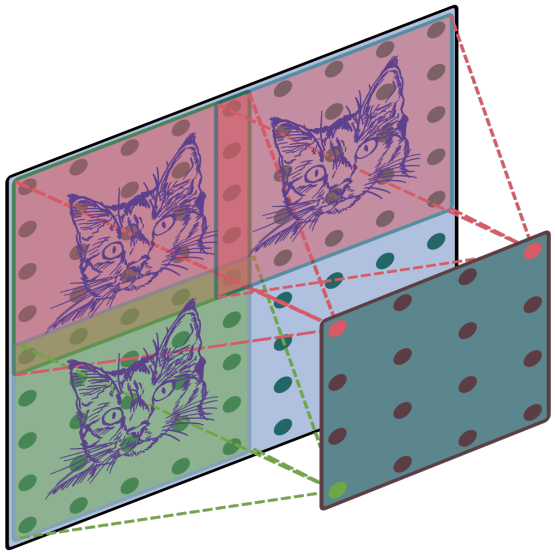

FIGURE 1.3: An illustration of the convolution layer concept and translation invariance property of CNNs. Each neuron in the convolution layer (second layer) applies the same feature detector (filter) on the input layer. Assuming that these filters detect if a cat exists in the image or not. Then, since they learn to detect the same feature, they will find the cat wherever its location (translation invariance).

the rigorous description of this concept was first established in the seminal work of LeCun et al [8].

In the following two sections, we first explain the basics of CNNs using image processing terminology to understand the underlying principles of CNNs. Then, we discuss their applications to NLP by giving an example that uses the sentence "Deep learning is awesome" on how the feature extraction mechanism works.

1.2.1 Convultional Neural Network: The Architecture

As mention above, the primary motivation beyond CNNs is to take advantage of the data structure. We, as humans, do that all the time to recognize different objects in images. CNNs mimics this ability. Therefore, instead of converting the image into a vector, we feed the neural network with a pixels matrix as shown in Fig.1.3. This representation preserves the data structure (in this case, the spatial structure of the data in the image). The main components in the architecture of a CNN, as shown in Fig.1.2, are

1. Input layer: this is the first layer of the CNN where it is designed to take advantage of the data structure. For example, if our CNN is designed to classify if black/white image is for a cat or a dog, then the input can be a square matrix ($n \times n$ neurons) where each neuron has a value that varies on scale $[0, 1]$, where 0 for white color and 1 for black color. In Fig.1.2, the input layer is a matrix with size 128×128.

2. Hidden layers: these are the layers between the input and output layers. They are trained to discover different features in the input (feature mapping), and then condensing the information about detected features using pooling, implying that there are two important sub-hidden layers:

   (a) Convolution layer: this layer may consist of several copies to detect different features (feature maps). In Fig.1.2, there are 6− (64×64) feature maps. The feature map relies on the *local receptive fields* concept: each neuron applies the same filter at a local region of the previous layer aiming to extract a specific feature. Fig.1.3 shows an example on (4×4) feature map trained to learn cat face. The neurons in this feature map use (5×5) filter, implying that the filter adjusts 25 weights and 1 bias through the training process. All the neurons in the feature map share these weights and bias, allowing recognizing the feature in any position within the image (translation invariance property -see Fig.1.3). In simple words, convolutional layer may consist of several scanners. Each of the learn to detect some feature.

   (b) Pooling layer: these layers are usually used after the convolutional layer, where a pooling layer follows every feature map. The CNN, shown in Fig.1.2, has 6−(32×32) pooling layer. The main objective of this layer is to simplify and condense the information in the output from the convolutional layer. For example, in Fig.1.2, each neuron in the pooling layer makes a summary of a (4×4) local field in the convolutional layer. There are several methods to summarise information, such as *max pooling* or L_2 *pooling*. According to the max-pooling method, the corresponding neuron in the pooling layer copies the maximum activation from its local convolutional field. This, in turn, leads to the loss of some information about the feature. For example, in Fig.1.3, the information concerning the exact position of the cat might be lost. However, the approximated position remains encoded by the neurons in the pooling layer.

3. Final layer: this is a fully connected layer, where each neuron is connected to all neurons from the previous max-pooled layer. The design elements of this layer, such as the number of neurons, depends on the performed task. For example, if the task is to recognize the handwritten digits, the layer contains ten neurons where the activation of each neuron represents the probability of detecting the corresponding digit, as shown in Fig.1.2.

1.2.2 Convolutional Neural Network for NLP Tasks

Instead of image pixels, the input to most NLP tasks is sentences or documents. These inputs are converted into numerical representations (matrices) using word embedding methods discussed in chapter 4. In the case of sentences, each row of the matrix, as shown in Fig.1.4, corresponds to a word.

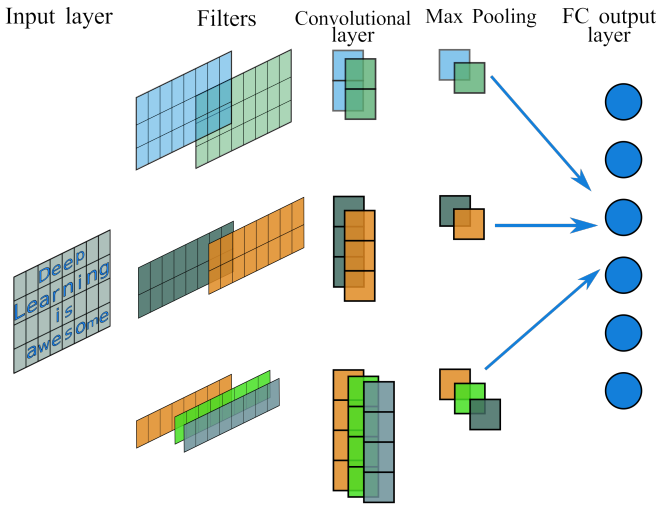

FIGURE 1.4: An illustration of a convolutional neural network for NLP tasks. After embedding the input sentence, a convolutional layer applies several filters to detect different features in the sentence. Then, a max-pooling layer obtains a compact version of the convolutional layer's output and sends it to a fully connected layer to be trained on some NLP tasks.

In image recognition problems, the filters applied by the convolutional layer can have any size (typically, their dimensions are 3×3 or 5×5). However, in NLP tasks, filters usually have a width equal to the input matrix's row length, while its height can extend over several words ($2 - 5$ words is a standard choice).

Fig.1.4 shows an example of CNN architecture for NLP tasks. The sentence "Deep learning is awesome" is first embedded into a numerical representation. Then, a convolutional layer applies filters of different sizes to detect different features (feature mapping). This layer is followed by a max-pooling layer that condenses the information received from various filters. Then, the information flow through a fully connected layer that performs some NLP tasks, such as film review classification, relation extraction, etc.

Chapter 2

BioCreative Challenges

According to BioCreative road-map, declared on the organizers website "the BioCreative (Critical Assessment of Information Extraction systems in Biology) challenge evaluation consists of a community-wide effort for evaluating text mining and information extraction systems applied to the biological domain" [9]. Aiming to achieve their goals, they, since 2003, called for several challenges: BioCreative (I-IIV).

Each challenge consists of several tracks. For example, the challenge BioCreative IIV, proposed in 2021 involves 5 tracks, including Track-1 (Text mining drug and chemical-protein interactions (DrugProt)) and Track-2 (Automatic extraction of medication names in tweets). In this master thesis, we use the corpus, proposed by BioCreative IIV for Track-1, to extract relations between chemicals and genes / proteins. Moreover, we are participating in Track-1 challenge aiming to the same objective, where each team is allowed to submit five runs of their best models.

In 2017, the BioCreative organizers proposed the challenge BioCreative VI. The Track-5 of this challenge uses the ChemProt corpus, a manually annotated corpus, where domain experts have exhaustively labeled the chemical and gene mentions, and the binary relationships between them [10]. For the Track-5 challenge, the organizers asked the participants to propose models able to extract 5 chemical-protein interactions (classes), that are relevant for precision medicine, drug discovery, and basic biomedical research.

The best ChemProt system was proposed in Ref.[11], where the authors obtained an F-score of 0.6410. They proposed 2 ensemble systems combining the results of 3 models: a majority voting system and a stacking system. These systems used support vector machines (SVMs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs).

The first ensemble system selects relations that were predicted by at least 2 models as a correct prediction. If there is no majority voting (at least two models) for a relation, then the "negative relation" class is assigned. The second system uses a meta-model trained on the predictions of the base models (SVMs, CNNs, and RNNs). In this system, they use random forest as a classifier.

Each model in the previous two systems exploits different type of features. The SVM features are the words surrounding the chemical and protein entities of interest, the bag-of-words between the chemical and protein entities in a sentence, the existence of a keyword between two entities, often manually built, and the shortest-path between the entities.

The CNN features are the original sentence and the shortest path between the two entities. The words in each sentence input of CNN models are represented by their embeddings concatenated with the position embedding, and the distance of each word from the entities.

Finally, in RNN models, the authors of Ref.[11] concatenate the word embedding with the part-of-speech, IOB-chunk tag, and two position embeddings.
Another promising architecture: the long short term memory models (LSTMs) were also proposed for this Track in Ref.[12]. The proposed model achieved an F-score of $0,5181$, which is smaller than that reached in Ref.[11]. These LSTM models consist of 3 to 6 bidirectional LSTM layers, followed by a fully connected layer, and an output layer with *softmax* activation. The inputs to each LSTM layer were obtained from the shortest dependency path between the chemical and gene entities in a sentence.

In this master thesis, we are going to employ both CNN-based models and transformer-BERT (Bidirectional Encoder Representations from Transformers) based models to study relations extraction of 14 classes between chemicals and proteins in DrugProt corpus, discussed in section 5.1.

Chapter 3

Relation Extraction and Word Embeddings: An Introduction

In this short introductory chapter, we start by a brief review of the relation extraction task. Then, we discuss the concepts of word and positional embeddings, one of the building blocks for any successful language model used by a neural network to perform a relation extraction task.

3.1 Relation Extraction (RE)

Relation extraction is a sub-task of information extraction aiming to find semantic relations between two identified entities (e_1 and e_2) in a given text from predefined relations. For example, these two entities, whose relation to be extracted, can be person-city, person-person, etc. The extracted relations can be employed to build *knowledge graphs*, commonly used by search engines [13]. For example, the following sentence from SemEval-2010 dataset [14]:

"He had chest pains and <e1>headaches</e1> from <e2>mold</e2> in the bedrooms."

is an input to RE task. The task output should be the following binary relation:

Cause-Effect(e2,e1)

In this example, we only pay attention to RE binary relations: classification task between only two entities. This example shows that the entities are ordered.The first entity is $e2$ and the second entity is $e1$. This order is important to the system in order to predicate the correct relation. In our work we are also interested in binary relations, where we have a dataset with 14 predefined relations (classes).

Many supervised approaches (see Ref.[15] and the references therein) are applied to achieve RE tasks, such as features-based and kernel-based approaches [16] . Features-based approach uses a set of features that are selected after performing textual analysis. Kernel-based approach requires preprocessed inputs in the form of parse trees [17] or bag of features kernels [15].

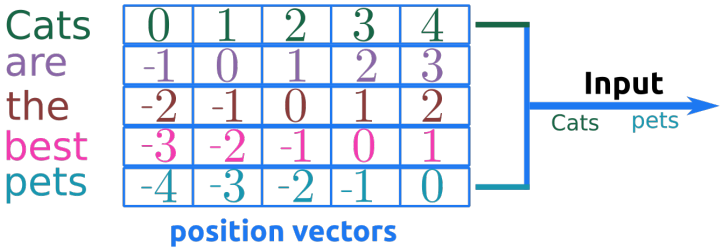

FIGURE 3.1: An example of positional embeddings for the sentence "Cats are the best pets". The position vectors of both words "Cats" and "pets" are needed when the objective is to extract their relation.

Deep learning (DL), a machine learning sub-branch discussed in chapter 1, can be also used. DL techniques are employed widely in the RE task.

Convolutional neural networks (CNNs), a sub-class of neural networks, were one of the first DL methods proposed to perform RE task in Ref.[17]. The authors proposed a model achieved representative progress without any external resources or NLP modules. This success led researchers to propose CNN-based models for the RE tasks, such as the multi-window CNN model suggested in Ref.[18]. However, CNNs can not capture temporal patterns , like treating data with long dependencies, that is the distance between the two concerned entities is long[16].

The previous limitation corresponding to the temporal patterns is solved by employing recurrent neural networks (RNNs) which have a kind of short internal memory sensitive to the last parts of the inputs [16]. Despite the success achieved by RNNs and CNNs in the RE tasks, they fail at capturing long context, a problem addressed by the state of art BERT model [19] based on transformer model, first introduced in the famous paper titled *"Attention is all you need"* [20]. In this paper, the authors introduce an attention mechanism allowing each word in the sentence to capture more contextualized content depending on its meaning and its relations with other words (we detailed this idea in chapter 4). On the other hand, the authors of Ref.[19] employed the attention mechanism to build a more "aware" language model (see chapter 4 for detailed discussion).

In the following chapters we are going to briefly review CNNs, their applications to NLP tasks, and then discuss in details the core ideas of BERT model. Then, we present some results obtained using BERT and CNN models.

3.2 Word and Positional Embeddings

Word embeddings is a known term in NLP gives vector representation for words. There are several methods used in word embeddings.

They can be divided in two large classes. The first class is the free-context embeddings in which the representations of words are independent in the sense that they do not contain any context content. One-hot vector [3] and term frequency-inverse document frequency (TF-IDF) [21] are examples of this class. The second class is distributional similarity based representations (DS-R) [22]. In methods corresponding to this class, the word embedding contains some information about the word, such as its meaning and interactions with other words in the text (contextualized representation). Contextualized representation allows similar and related words to have similar representations, which can be used as input features to neural network models.

All DS-R methods requires training a neural network to obtain the word embeddings. Word2Vec and GloVe are classical examples of the DS-R methods. However, state of art models that outperforms Word2Vec and GloVe emdeddings have been suggested. This includes Elmo [23], transformer encoder [20], and BERT [19] models.

Learning words embeddings from scratch is not only a time consuming task, but is also associated to the availability of the corpus in the corresponding domain. Therefore, pre-trained word embeddings are used as input to NLP models. Selecting the pre-trained word embeddings is an important step to build accurate models.

In addition to word embeddings, another concept plays a key role in RE task: positional embeddings (PE). As detailed in Fig.3.1, PE is simply represented by a vector containing the distance of each token to the different entities in given text . In the figure, we show the position vectors for the different words in the sentence:

"Cats are the best pets."

Now, if one wants to train a neural network to extract the relation between "Cats" and "pets", this NN is then fed with the position vectors of these words, as shown by the blue arrow in Fig.3.1. The previous sentence can be represented with different embedding methods explained in this section. However, predicting the relation between the two words "Cats" and "pets" can be strongly affected by the embedding methods. For example, the one-hot vectors representation leads to orthogonal input vectors, implying that the different words in each sentence are independent. On the other hand, the embeddings obtained by BERT model can capture rich representations for these words compared to other methods (see chapter 4).

Chapter 4

BERT Model: "Attention is all You Need"

In this chapter, we will discuss the details of the state of art bidirectional encoder representation from transformers (BERT model [19]) and its different aspects related to natural language processing (NLP). This chapter will lay out the essential ideas beyond the model. We start by introducing the general idea about the model. After that, we briefly discuss sequence to sequence models based on recurrent neural networks and the short memory problem that, besides the aim to obtain precise contextualized representations, motivates the transformer model (a language model equipped with an attention mechanism). Then, we present the details of the attention mechanism and the architecture of the transformer, which is composed of encoding and decoding parts (transformer encoder and transformer decoder). The transformer encoder generates a context "aware" representation of the words, while the transformer decoder may perform some natural language tasks, such as translation. Since the BERT model makes use of the transformer encoder only, we focus on it. Finally, we add together all BERT building blocks and discuss the two strategies used in training and fine-tuning the model to perform NLP tasks.

4.1 Introduction

Bidirectional Encoder Representations from Transformers (BERT) is a model proposed by researchers at Google AI Language in 2018 [19]. BERT is the first model to apply bidirectional training of transformer to language models, where previous models looked at the input sequence from left to right or combine the left-to-right and right-to-left training (unidirectional models). The bidirectional training helps the model learning the contextual relations between words (or sub-words) in a text even if the words are far apart from each other. This implies that every word in the sentence will pay attention to the relative importance of the other words, meaning that this algorithm has a kind of self-attention which is, in fact, the main contribution transformers brought to the NLP domain [20].

When estimating the prediction efficiency, most unidirectional models are trained to predict the next word in a sentence, limiting the context content of

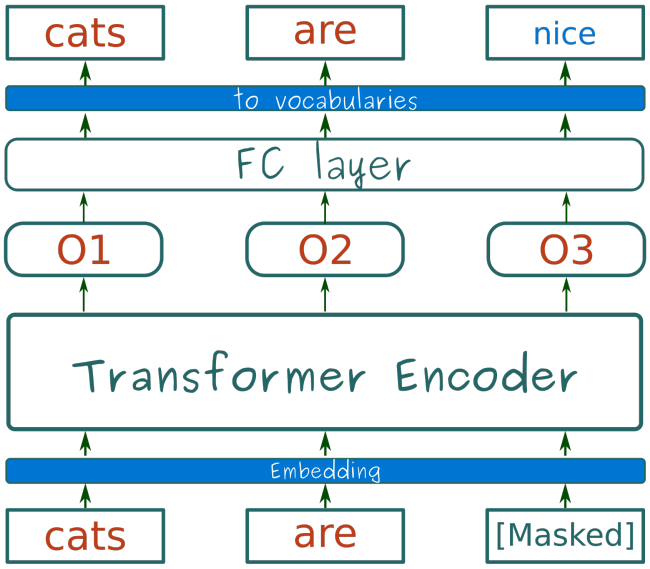

FIGURE 4.1: A simple visualization of BERT model. The sentence is first encoded using an embedding algorithm and then passed to the transformer encoder once (bi-directionality). The transformer's output is a compact representation of the sequence whose tokens now have strong attention to each other even if the distances separating different words are large. This representation is then used as an input to a fully connected layer that predicts the masked tokens. Finally, another embedding layer is added to convert the numerical representation of words back again into vocabularies.

the learning process. Therefore to overcome this difficulty, BERT uses two training strategies: the masked language model (MLM) and the next sentence prediction (NSP). In the first strategy MLM, BERT model takes as input sequences of words, where 15% of the words in each sequence are replaced with a [MASK] token, the model then attempts to predict the masked words. In the second strategy NSP, the input is two sentences. The goal of BERT then is to predict whether the second sentence is a subsequent of the first one or not. One of the most powerful points about BERT is that the training is performed simultaneously for both strategies where the objective is to minimize their combined loss function. In addition, it also supports a massively parallel treatment of the data, which saves resources and make the method more attractive compared to other models.

The main components of the BERT model, as shown in Fig.4.1, are

1. An embedding layer at the bottom level. This layer maps the input sentence, with some words masked, into an embedded vector involving some context. For example, a clever embedding algorithm, such as the stat of art "Embedding from Language Models" (ELMo) algorithm [23], can be used. ELMo gives the word an embedding representation based on its meaning in the context and other contextual information, such as its interactions with other words. As shown in the figure, some words are replaced by [MASK] token aiming to train BERT model on predicting the masked words.

2. A transformer encoder which generates "deep contextualized word representation" of each input vector (the transformer encoder block in Fig.4.1).

3. The output of the transformer encoder is used as an input to a fully connected NN (classification layer) to predict the masked words (MLM strategy) and the relationship between two sentences (NSP strategy).

4. The output of the previous step is now converted back into vocabulary representation with the masked words replaced by the predictions of the model.

Before going further, we briefly review the Sequence to Sequence model since the underlying principles (recurrent neural networks and encoders) will repeatedly appear in this chapter. This review will shed light on these models and their disadvantages that motivated the state-of-art transformer and BERT models.

4.2 Sequence to Sequence Model

Sequence to sequence models are DL approaches used successfully in many NLP tasks, including language translation, text summarizations, conversational models, word prediction, etc. The architecture of these models is based on recurrent neural networks (RNNs).

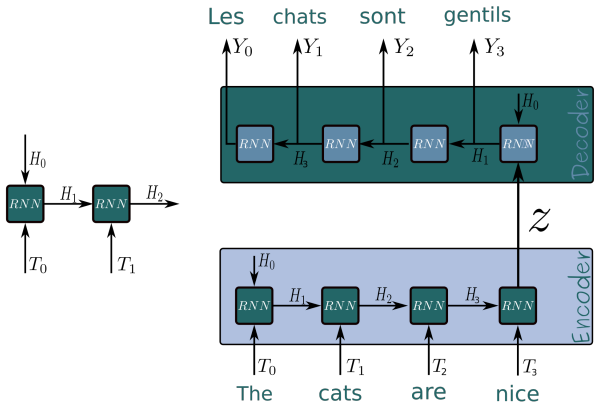

FIGURE 4.2: The concept of RNNs is shown on the left frame, where the output H_1 of the network is encoded as a hidden state of the NN and passed, with the next token T_1, as the following input to the RNN. This hidden state represents the memory of the NN, that is, the effects resulting from the previous tokens in the sequence. On the right frame, we show an illustration of the Seq2Seq model. Here, the encoder (at the bottom) generates the context vector z of the English sentence, which is then translated by the decoder into French.

RNNs are considered as a generalization of the feed-forward NNs that have internal memory. This makes them suitable for processing sequential data, including data collected at different timesteps or sentences, which can be represented as a sequence of tokens converted to numerical representations using some embedding algorithms. RNN performs (repeats) the same mapping process for each input of the sequence. The output corresponding to the current input depends on the previous input, represented by the current hidden state of the RNN (its memory), as shown in Fig.4.2 . This dependency implies that RNN has a memory that is somehow short and limited to relatively short sequences. Still, they perform very well for sequences in which the active token depends strongly on the previous one. However, there are some drawbacks associated with RNNs. These disadvantages include the gradient vanishing problem, the difficulty of training, which will be a slow process since the backpropagation algorithm is repeated at each step, and its inability to process very long sequences with *relu* or *tanh* functions.

To overcome some of these drawbacks, such as the gradient vanishing and short memory problems, Long Short-Term Memory (LSTM) networks were designed. They are updated or modified versions of the RNNs and are suitable for the same tasks mentioned above. LSTMs learn to keep only the relevant information when predicting some NLP tasks. However, they need more computational resources compared to RNNs, and are also more challenging to train.

Seq2Seq model is built above these models. Its main objective is to map an

input sequence to an output one by first generating a representation of the input and then converting it into another sequence. This process includes two layers: the encoder and the decoder. These layers are stacked RNN layers, such as LSTMs (see [24] for a nice discus ion). The objective of the encoder is to generate a compact representation of the input (also known as context vector), while the decoder maps this vector into an output sequence.

For example, the input of the encoder can be the following sentence:

"The cats are nice."

and the output of the decoder is its French translation:

"Les chats sont gentils."

This encoding-decoding process of the sentence is shown in the right frame of the Fig.4.2. It starts with the encoder generating an intermediate representation, which is then mapped into its corresponding French translation by the decoder.

The short memory in this type of NNs results from the context vector's inability to capture the information coming from all the tokens in long sequences, leading to an NN that always pays more attention to the last parts of the sentence. This is also applied to the relationships extracted between different words in a sentence or even between different sentences. In this case, every word will appear more connected to its direct neighbours. However, it is worth mentioning that the model won't face any problem when dealing with short sentences like the previous example.

Obviously, the addition of any attention contents to the words' representations will further improve any models intended to implement NLP tasks. This is the subject of the next section.

4.3 Pay Attention to Attention

Obtaining the best representation of different words in a sentence is of crucial importance for any NLP task. As much as the representation of a specific word is "*aware*" of its meaning and relationships with other words, as much as the model will be more efficient and natural since it mimics the way we recognize the relationships between different words in a sentence.

For example, let us look at the sentence in the top frame of Fig.4.3. It is easy for us to recognize the relationships between the words "Paris", "Eiffel", and "France". The same is true for "She" and "Born". However, the situation is more complicated for models whose memories are short and consider the last parts of the sequence as the most relevant ones. Therefore, an algorithm that can capture the relationships shown in Fig.4.3 will give better encoding (representation) of the sequence. This representation takes into account the context of every word, and therefore, gives better results when used as input to another fully connected NN trained to performs some NLP tasks, such as

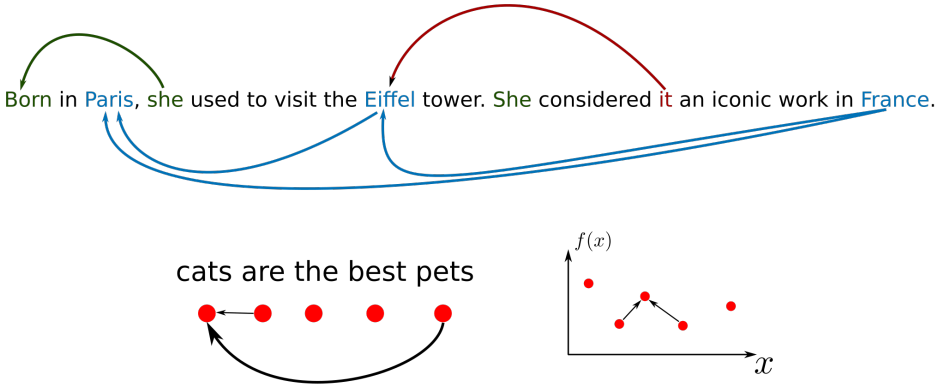

FIGURE 4.3: In the top frame, we give an example showing our self-attention of the relationships between words in a sentence. In the left-bottom frame, we show another example indicating that the most important relationship for "cats" is with the last word in the sentence "pets". However, suppose one follows the same line of thought applied for interpolation, as shown on the right-bottom frame. In that case, we obtain that the interpolated value of a function at specific point is strongly related to its neighbours, that is, as far as the neighbour is, as weak as its effect on the interpolated value.

review classifications. The generation of such a "*highly-aware*" representation is the main contribution of the transformer encoders: the building blocks of BERT.

Let us now take a closer look at the attention mechanism. We are going to present the following three methods:

- Embedding-based attention.
- Self-attention mechanism, first introduced in "Attention is all you need" paper [20].
- Multi-head attention: an improvement of the previous mechanism that allows the representation to include more contextual contents [20].

4.3.1 Embedding-based Attention

This attention mechanism depends mainly on the method used in embedding words. Suppose the algorithm accounts for some context for each word, such as its meaning and relationship with others. In that case, it is possible to apply a linear transformation leading to a more contextualized representation of each word. This method is deterministic in the sense that there are no weights to learn. The steps for performing this attention method are

1. Apply an embedding algorithm which allows involving some information about interactions between different words. In the figure, every word in the sentence "Cats are the best pets" is embedded to the corresponding vector. For example, "Cats" is given by x_1, which can have

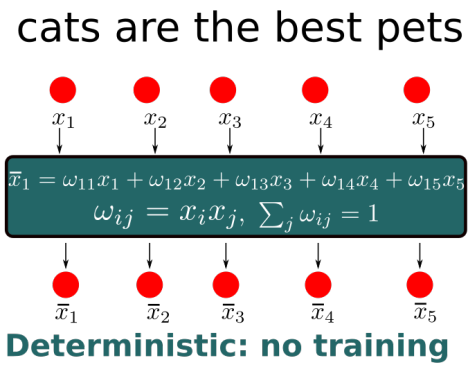

FIGURE 4.4: The mechanism of embedding-based attention. In this deterministic attention encoding, one relies on the word embedding algorithms used to convert the sentence (here, "Cats are the best pets") into numerical vector representation used by the NN. If every word in the sentence is given some contextual meaning, the dot product between different vectors $(x_1, x_2, \cdots, x_5)$ captures the relevant importance of relationships between specific words.

any length depending on the chosen algorithm (e.g. the length of each token is 768 in BERT base and 1024 in BERT Large). This step is shown at the top of the Fig.4.4.

2. Calculating different weights: in this step we calculate the weights (relative importance) between different words by simply taking the dot product of their vector representations. For example, in Fig.4.4, the relationship between the i^{th} and j^{th} words is given by

$$\omega_{ij} = x_i x_j, \tag{4.1}$$

For the words "Cats" and "pets", it reads ω_{15}. If the embedding algorithm is good enough, then one expects

$$\max_{i \neq j}\{\omega_{ij}\} = \omega_{15} \tag{4.2}$$

However, even it's often not the case, if one starts with arbitrary embedding that respects some mathematical rules, then $\omega_{i \neq j} = 0$. Therefore, in this case, no attention is taken into account.

3. After obtaining the weights, one applies a linear transformation for each vector, as shown in Fig.4.4. For example, the "Cats" word, represented by x_1, is transformed into $\bar{x}_1$ which now contains more contextual meaning.

4.3.2 Self-attention Mechanism

Following the previous section, one question comes to mind: *can we further improve the attention content of the word, or in general a token, representation?*

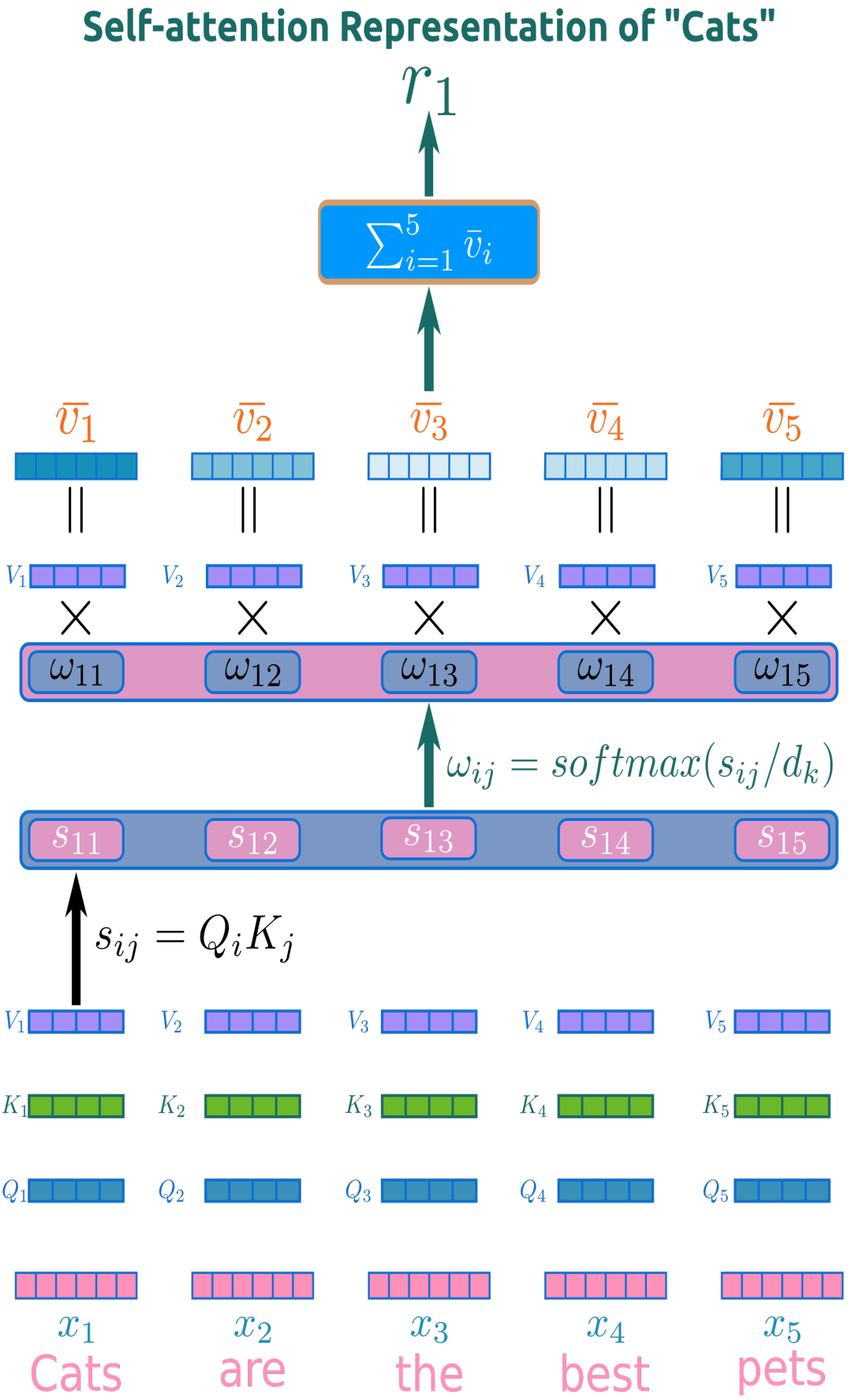

FIGURE 4.5: An illustration of the self-attention mechanism applied to first word in the sentence "Cats", given by the input vector x_1. The resulting representation of x_1 is r_1.

The answer turns out to be sure we can! This is, in fact, the breakthrough transformer model achieved by introducing the concept of self-attention [20]. In this mechanism, each token is associated with three new vectors: query, key, and value.

To understand the advantages of these vectors, let us consider a "naive" version of the search process on the internet. If you want to search for the word "Cat", then this is your query. Then, a correspondence between this query and different search results is established. These results can be considered as the components of the key vector. Finally, for each of these results, there exists a number that reflects the strength of the relationship between the query and the corresponding result. These different numbers associated with different results are the component of the value vector.
For our sentence example,

"Cats are the best pets"

we can summarize the subsequent steps of the self-attention mechanism (self-attention layer), detailed in Fig.4.5 as follows

1. An embedding layer converting each word of the sentence into numerical vector which contains some context. For example, "pets" is transformed into x_5. Let the length of the vector, that is the number of columns, is n.

2. For each word, say "pets", we associate three vectors of the same length (k). These vectors are the query, the key, and the value. For the word "pets", they are Q_5, K_5, and V_5, respectively. One then can apply the search analogy explained above, that is, the indicator of the relationship (hereafter, the *score*) between the i^{th} and j^{th} words is expressed as the dot product between the corresponding query Q_i and key K_j, as given by s_{ij} in Fig.4.5, which shows the self-attention representation for the word "Cats".

3. As soon as the scores are calculated, we then normalize with respect to the square root of the length of the vectors leading to more numerically stable algorithm. Then, we obtain the weights by normalizing the scores with the *softmax* function, implying that $0 \leq \omega_{ij} \leq 1$, and $\sum_j \omega_{ij} = 1$. This step is shown in the middle part of Fig.4.5 -again for the word "Cats".

4. After obtaining different weights for each token, we simply multiply each of them with the corresponding value vector to estimate the different relationship vectors ($\bar{v}_i$). In Fig.4.5, we plot this process at the top part of the figure for "Cats" word. As darker as the color of $\bar{v}_i$ vector becomes, as stronger as the corresponding relationship. For example, it is shown in the figure that $\bar{v}_1$ and $\bar{v}_5$ have the darkest colors, meaning, as expected, that x_1 ("Cats") interacts more with itself and x_5 ("pets").

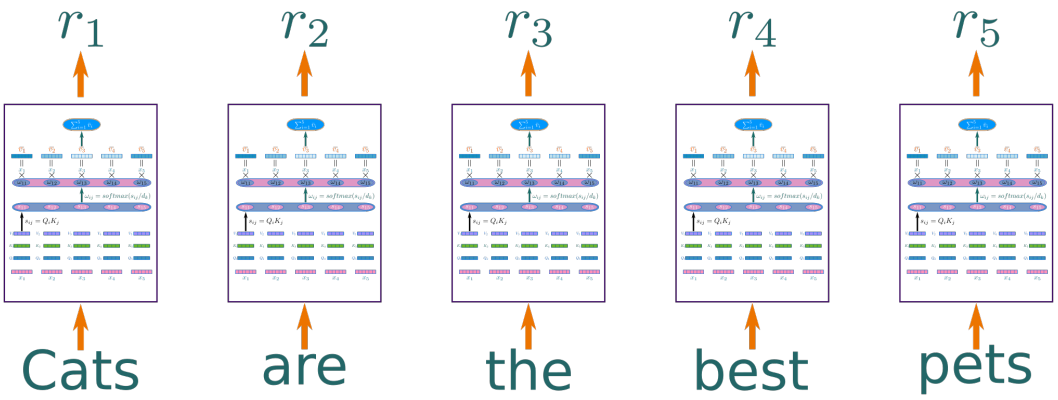

FIGURE 4.6: This figure shows how the same self attention mechanism, detailed in Fig.4.5, is repeated for all the words in the sentence "Cats are the best pets".

5. The last step is to output the representation of the word obtained by the self-attention layer. This is done by adding together the different relationship vectors ($\bar{v}_i$) leading to the representation r. In the figure, r_1 is the encoded representation of the word "Cats" after the information flowed through the self-attention layer.

The previous steps are applied to each token in the sentence. Fig.4.6 shows this process for sentence example, where every small box in the figure is an identical copy of Fig.4.5.

Until now we did not discuss how to calculate the components of Q_i, K_i, and V_i (self-attention vectors). In fact, here is comes the beauty of the attention model, which suggests that they can be calculated by multiplying the input vectors with corresponding matrices $\mathcal{M}_Q$, $\mathcal{M}_K$, and $\mathcal{M}_V$, respectively. This process is detailed in Fig.4.7 for our example. The elements of these matrices (weights) will be adjusted by training the model. As soon as these weights have been estimated, one can calculate the self-attention vectors by multiplying the input matrix with the corresponding self-attention matrix, as shown in Fig.4.7.

Can we improve this great idea further? Again the authors of "Attention is all you need" [20] affirmed that the answer is yes, we can. To do so they introduced the idea of multi-head attention, which are going to discuss in the next section.

4.3.3 Multi-head Attention

Following the self-attention mechanism detailed in Fig.4.5 and the previous section, we were able to obtain more contextualized representation (e.g. r_1 for "Cats") compared to that of the embedding algorithm. However, as shown in Fig.4.5, the strongest relationships of "Cats" are with itself and the

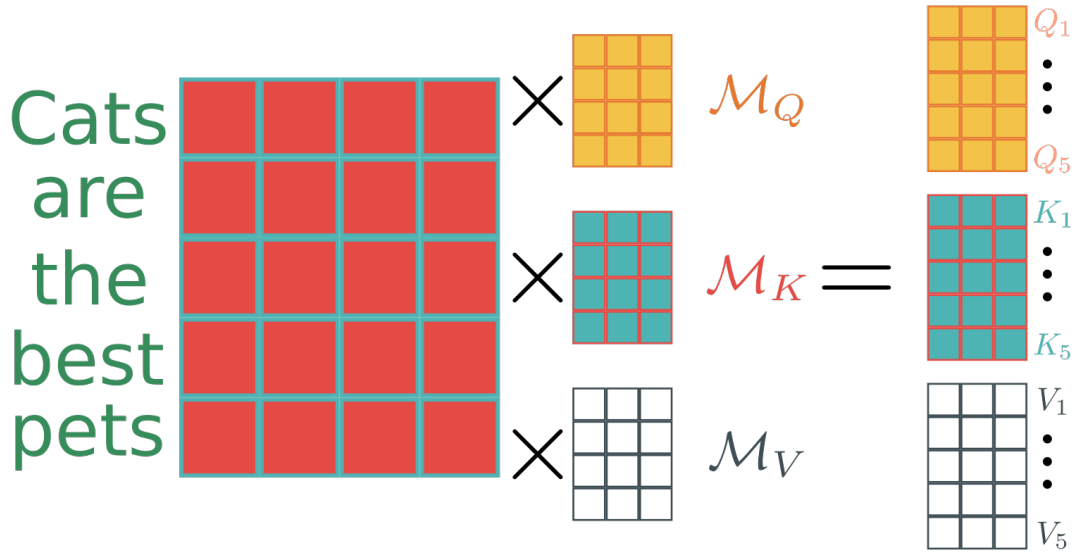

FIGURE 4.7: An illustration of the calculation of different queries, keys, and values vectors for the the words in our example "Cats are the best pets". The weights of $\mathcal{M}_Q$, $\mathcal{M}_K$, and $\mathcal{M}_V$ matrices are learned by the training process, where initial values are randomly chosen.

word "pets". Therefore, this kind of attention mechanism can lead to a *"self-dominated"* representation, and the question of improving it becomes relevant. On the other hand, having different representations, depending on various criteria, such as the meaning of the word and its relationships with others, is also attractive since it allows encoding more information about the word in its numerical vector (the output of the attention layer). This is the main idea underlying the concept of *multi-head attention*, which was one of the pioneering ideas proposed and applied by the authors of "Attention is all you need" [20].

The core idea of multi-head attention is to have several copies of the self-attention mechanism, where each one forms a different head of the attention layer. We can understand this concept by looking back at our sentence example, "Cats are the best pets". The essential steps of this attention method are (see Fig.4.8)

1. At input level, each word in the sentence is embedded using some embedding algorithm which probably encodes some meaningful content for every word. In Fig.4.8, this step is shown on the left of the top frame ($5 \times n$ red rectangle, for example $n = 768$ for BERT base and $n = 1024$ for BERT large).

2. Different copies of the input flows through different heads, where the matrices $\mathcal{M}_Q$, $\mathcal{M}_K$, and $\mathcal{M}_V$ are randomly initialized with different values for each head. In Fig.4.8, we give an example on $4-$headed attention layer with a distinguish color for each head. The elements of

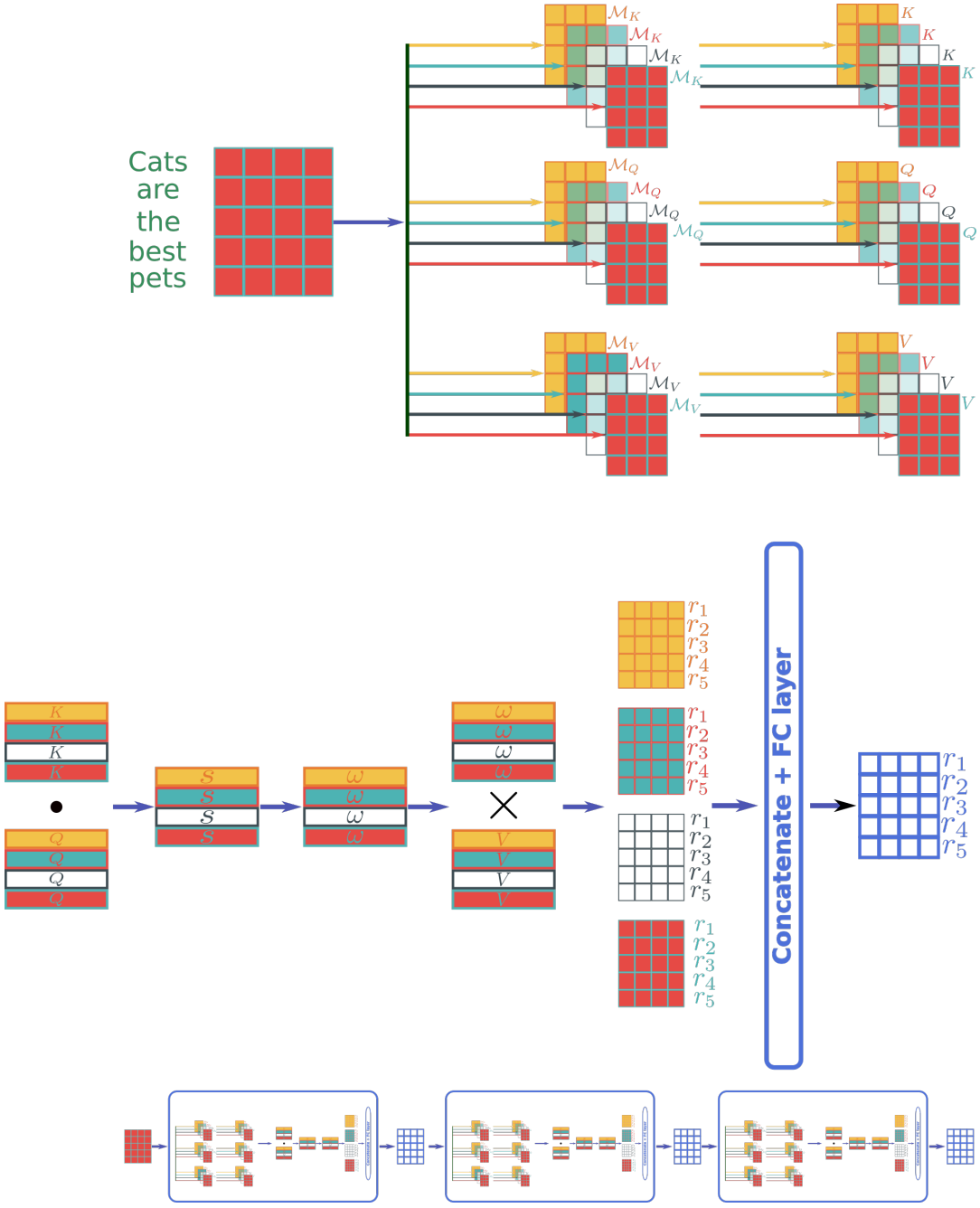

FIGURE 4.8: A visualization of the mechanism of multi-head attention layer using four heads in the top and the middle frames. In the bottom frame, we plot a stack of multi-head layers (the details are explained in the section 4.3.3).

attention matrices are obtained by training the NN, and then used to obtain query (Q), key (K), and value (V) vectors for each word.

3. Similarly to self-attention mechanism, we calculate the scores in each head by taking the dot product of the corresponding K and Q vectors, which are shown in the same color in every head in middle frame of Fig.4.8.

4. We scale and normalize the scores in each head using *softmax* function to obtain the the corresponding weights, as shown in the middle frame of the figure.

5. For every word in every head, the weights are multiplied with value vectors of different words and then added together to obtain the word representation in that head. This step is shown in the middle frame of Fig.4.8, where we see four representation for every word with distinguished colors related to the relevant head.

6. Finally, the representations resulting from different heads are concatenated and passed to fully connected layer leading to a hybrid representation that incorporates more information about every word, as shown on the right of the middle frame of Fig.4.8.

7. The previous steps constitute an encoder. One can then stack many such encoders next to each other, and the output of one encoder will be the input of the next one, as shown in the bottom frame of the figure. There are 12 layers of these encoders in BERT base, while they become 24 for BERT large.

4.4 Transformers

This section adds together the different ideas discussed in the previous sections to define the transformer: a model that uses attention mechanisms, such as the multi-head attention method, to accelerate the training process. Transformer consists of encoding (transformer encoder), as shown on the right frame of Fig.4.9, and decoding (transformer decoder) parts. The transformer encoder is a stack of subsequent encoders aiming to transform the input into a better representation that flows through the transformer decoder composed of a stack of decoders dedicated to performing some NLP tasks, such as translating a sentence from English to french, evaluating the reviews, etc.

As shown on the left frame of Fig.4.9, the encoder consists of two sub-layers. The first is the multi-head attention layer discussed in section 4.3.3, while the second is a feed-forward NN. The decoder, which won't be discussed further since BERT model uses only the transformer encoder, has three sub-layers. The first and the third ones are identical to those in the encoder. However, the decoder has an intermediate layer that draws attention to some relevant words of the input sequence.

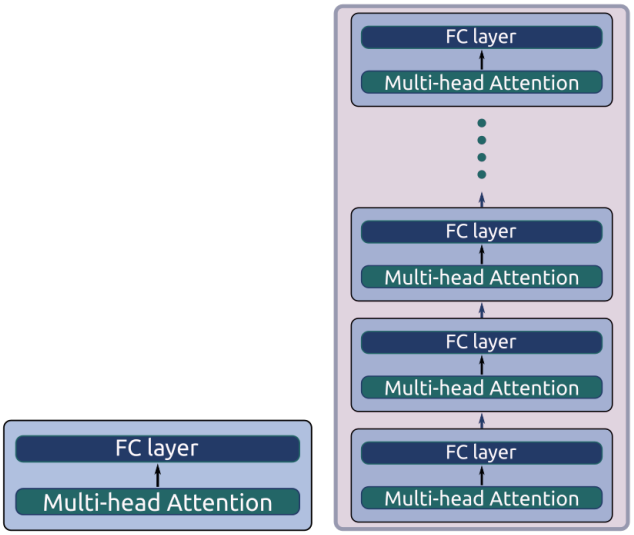

FIGURE 4.9: On the left, we show a high-level illustration of the encoder, detailed in section 4.3.3. On the the right frame, we plot the transformer encoder used in BERT model, which is represented by the block transformer in Fig.4.1. As mentioned before, in BERT base this transformer consists of 12 encoders, while in BERT large it involves 24 encoders.

As shown in Fig.4.1, at the bottom level of the BERT model, the sentence is transformed into the numerical vector using some embedding algorithm. Then, the input is passed to the transformer encoder, where it flows through the subsequent internal encoders (12 for BERT base and 24 BERT large). Each internal encoder gives an output with better contextual content of the word, which is the main idea over which BERT model is designed. It is worth mentioning that the second sub-layer (the feed-forward NN) in each encoder is not aware of the dependencies between the different representations resulting from the attention layer of the words. This property implies that the inputs to this sub-layer can be executed in parallel. The transformers, therefore, are not conceptually attractive only but also computationally since they allow a massively parallel training of the model.

The final output of the transformer encoder is, fortunately, more representative of the vocabulary input than that resulting from the embedding algorithm. This property is a result of encoding more contents, such as the meaning of the word, its context, its interactions with other words, and the relative strengths of these relationships. Therefore, using NNs with transformer encoder above the embedding layer will lead to more efficient models, which is the breakthrough the authors of BERT paper [19] achieved.

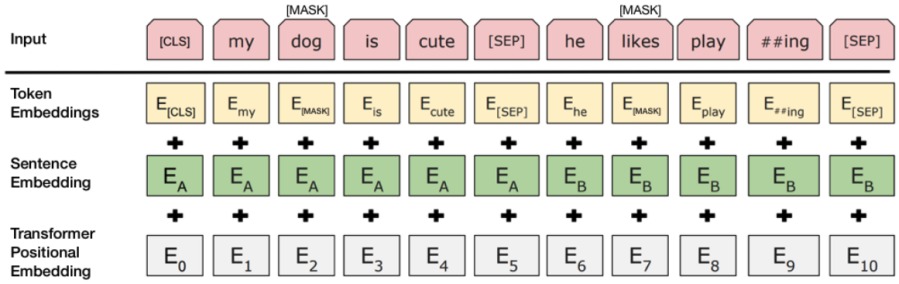

FIGURE 4.10: BERT input is the sum of the token embeddings, the segmentation embeddings and the position embeddings. source: BERT paper[19].

4.5 BERT Again: Training Strategies and Fine-tuning the Model

We can now collect the ideas discussed in the previous sections together to define the BERT model. Although we gave the general idea of the model in the introduction, it is good to recall it before going further to present the training strategies of the model. BERT model can be defined as an NLP language model that maps each word of an input sentence into highly aware context representation. The model performs this mapping by applying the bidirectional training of the transformer encoder. The transformer is composed of a stack of encoders, where each one contains two sub-layers: a multi-head attention layer followed by a fully connected layer.

As stated in section 4.1, the model consists of an embedding layer, followed by the transformer encoder. Then the representation obtained by the transformer flows through a fully connected NN trained to both predict the masked words in the sentences (in BERT paper [19], 15% of them are replaced by [MASK] token), and whether two sentences are after each other or not. As soon as the training on these two NLP tasks is accomplished, we can use the model as building block (transfer learning) of another NN architecture designed to perform other tasks, such as relation extraction, translation, text similarity, etc (fine-tuning). Transfer learning has proved to be helpful in different machine learning domains, such as image recognition. This concept includes using a pre-trained NN on some specific tasks to perform a new one. This idea is essential in BERT model (see [25] for a nice brief introduction).

As mentioned above, before sending the sentence through the BERT model, 15% of the words are replaced by [MASK] token. According to masked LM strategy, BERT attempts to predict the masked words based on the context content the transformer encoder provides to each un-masked word representation. As shown in Fig.4.1, to predict the masked words, the transformer output is passed to a classification layer trained to anticipate these words, and then calculating the occurrence probability of each word using *softmax*

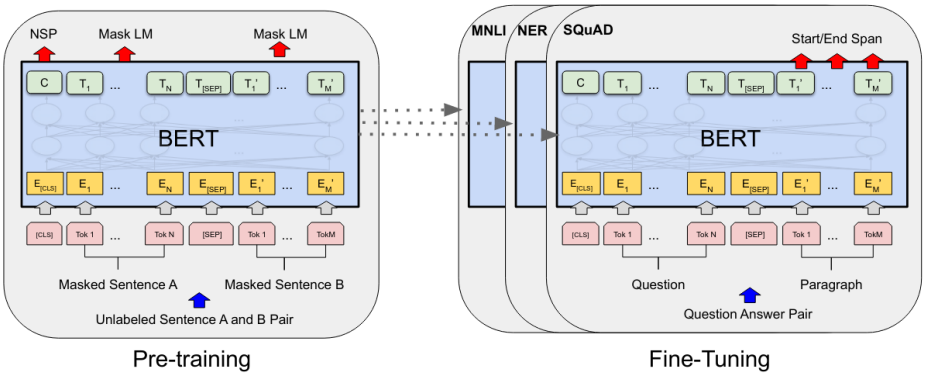

FIGURE 4.11: An illustration of fine-tuning pre-trained BERT model to perform other tasks. Notice that only the output layers of the original BERT model are modified to adapt to new tasks. source: the BERT paper[19].

function.

At the same time, the model is also trained using NSP strategy: predicting whether two sentences are subsequent to each other. Therefore, in this case, the model is fed with a sequence representing the two sentences. This sequence then flows through the transformer encoder. Finally, the FC NN located above the transformer learns to estimate the probability of the two sentences being successive to each other. The first sentence of each input starts with [CLS] token, while a [SEP] token is added at the end of every sentence. Fig.4.10, taken from the original paper [19], explains this procedure. As shown in the figure, each embedding consists of three parts: token embeddings which account for words embeddings, the sentence embedding allowing us to recognize the corresponding sentence of each token, and the transformer positional embedding indicating the position of each token in the input sequence. The optimization problem in this scenario of BERT model is to minimize the combined loss function of the two strategies: masked LM and NSP.

After training BERT model, the pre-trained model parameters are then used to initialize the model for performing different NLP tasks (fine-tuning BERT). This means that the essential architecture of BERT model, with all deep contextualized representations of words gained from the transformer, remains the same. Often, the fine-tuning concerns the output layers, as shown in Fig.4.11, taken from BERT paper[19], for the question answering task, the named entity recognition (NER), and the Multi-Genre Natural Language Inference classification task (MNLI). In BERT paper [19], the authors present BERT fine-tuning results for 11 NLP tasks, such as general language understanding evaluation (GLUE) and the question answering problem using Stanford Question Answering Dataset (SQuAD).

Chapter 5

Experiments and Results Analysis

In this chapter, we present the results of relation extraction experiments performed first using CNN model alone, then SciBERT model, and hybrid model (SciBERT + CNN) implemented in PyTorch framework, an open source machine learning framework that provides high-level API to implement deep neural networks. We also discuss preliminary results of the effects of employing positional embeddings and entity embeddings on the inputs of the models.

SciBERT model, a state of art model for RE task, is a variation of BERT model which uses 3.17 billion words of mixed scientific texts (82% of the words correspond to biomedical field) [26]. In previous works [27], this model was applied to ChemProt corpus, a corpus of relationships between proteins and chemicals, distributed in train, validation, and test sets [28].
In the experiments studied in this thesis, we use DrugProt corpus briefly introduced in the next section. These experiments were performed on Grid5000, "a large-scale and flexible testbed for experiment-driven research in all areas of computer science, with a focus on parallel and distributed computing including Cloud, HPC and Big Data and AI" [29].

5.1 DrugProt Corpus

DrugProt corpus was created by the BioCreative organization. It is a part of BioCreative VII (Track1 - Text mining drug and chemical-protein interactions [30]) . DrugProt dataset consists of three distinct groups: training, development, and testing sets.

The training set is composed of 3500 abstracts, 89529 annotated entities (CHEMICAL , GENE-Y, GENE-X), and 17288 annotated relations. We treated GENE-Y and GENE-X equally. We then use this set in training and testing our models, we split the training set into 80% for training and 20% for testing(see table 5.1). The development set consists of 750 abstracts, 18858 annotated entities (Gene), and 3765 annotated relations.

Finally, the testing set consists of 10750 un-labeled abstracts. Therefore, they were not used in evaluation phase of our models. However, the BioCreative VII organizers will use this set to evaluate the competitors participating

Relations	**Total**	Training	Testing
ACTIVATOR	1423	1149(8.15%)	274
AGONIST	658	524(3.72%)	134
AGONIST-ACTIVATOR	29	26(0.19%)	3
AGONIST-INHIBITOR	13	12(0.08%)	1
ANTAGONIST	970	767(5.44%)	203
DIRECT-REGULATOR	2240	1785(12.67%)	455
INDIRECT-DOWNREGULATOR	1328	1073(7.61%)	255
INDIRECT-UPREGULATOR	1376	1078(7.65%)	298
INHIBITOR	5377	**4307(30.57%)**	1070
PART-OF	882	729(5.17%)	153
PRODUCT-OF	916	735(5.23%)	181
SUBSTRATE	2002	1591(11.29%)	411
SUBSTRATE_PRODUCT-OF	24	14(0.099%)	10
no_relation	44932	300(2.129%)	300

TABLE 5.1: Classes distribution (13 classes + no-relation class):Training(14090),Testing(3748). The ratio appearing in the training column represents the percentage of each relation in our training set.

in the BioCreative VII Track-1 challenge aiming *"to promote the development and evaluation of systems that are able to automatically detect in relations between chemical compounds / drug and genes / proteins"*, according to the organizers [30].

The training and development examples were annotated by domain experts with chemical and gene entity mentions, and the relations indicating the biological interactions between the entities. There are 13 types of biological interactions to be considered in BioCreative VII:

$$INDIRECT-DOWNREGULATOR, INDIRECT-UPREGULATOR, DIRECT-REGULATOR, ACTIVATOR, INHIBITOR, AGONIST, ANTAGONIST, AGONIST-ACTIVATOR, AGONIST-INHIBITOR, PRODUCT-OF, SUBSTRATE, SUBSTRATE\_PRODUCT-OF \text{ or } PART-OF.$$

Table 5.1 shows the details of training and testing sets of the classes shown above, and that will be used to train CNN and hybrid (CNN+BERT) based models in sections 5.4 and 5.5, respectively.

5.2 Data Preprocessing

We pre-processed the training data to use it as input to our models. The pre-processing, as shown in Fig.5.1, steps are:

FIGURE 5.1: The header of the pre-processed file containing the inputs of our models as detailed in section 5.2

1- Sentence tokenization: split the abstracts into sentences using the sentence tokenizer from nltk (natural language toolkit from python), as shown in Fig.5.1.

2- Extracting entities and their indices in each sentence. If no entity exists or one of the two entities, whose relation to be predicted, is absent we set the index to "NONE". This extracting step is shown as "entity1", "entity2", "pos1", and "pos2" in Fig.5.1.

3- Labeling the sentences with one of the 13 relations, mentioned above, if it exists. Otherwise, we label it with the "no-relation", shown in relation columns of Fig.5.1.

Some sentences contain more than two entities. In this case, we generate different copies of the same sentence. Each copy is associated with one pair of the entities (CHEMICAL-GENE), where one should pay attention to the order, and not considering the relations GENE-GENE and CHEMICAL-CHEMICAL. For example, if the sentence has c chemical entities and g gene entities, it can be proved that the total number of generated copies (N_c) is given by the following simple equation:

$$N_c = c \times g \tag{5.1}$$

As soon as the N_c copies have been generated, we again apply the steps 2 and 3 shown above.

5.3 Position Embedding (PE)

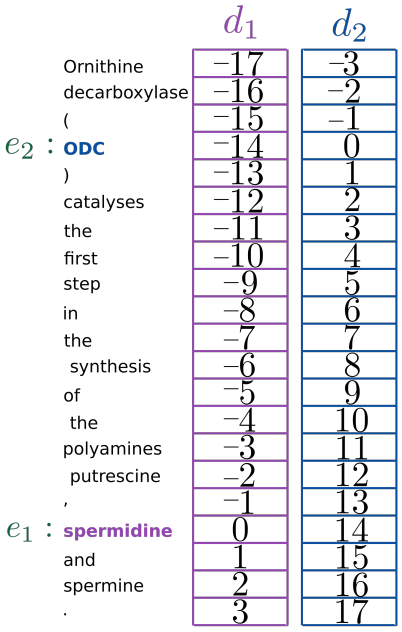

FIGURE 5.2: An illustration of positional embedding of the two entities "ODC" and "spermidine" in the given sentence.

Position embeddings, discussed in section 3.2, are generated based on the relative distances between tokens and the two entities, that we are interested in finding the relations between them. Fig.5.2 shows an example of finding the positional embeddings for the entities "ODC" and "spermidine" in the following sentence:

Ornithine decarboxylase (**ODC**) catalyses the first step in the synthesis of the polyamines putrescine, **spermidine** and spermine.

In the figure, d_1 represents the positional embedding of the entity "spermidine", where we first consider the entity to be at the origin (zero position). Then, we assign to each token an integer indicating its distance from the entity (e.g. the distance separating the token "of" from "ODC" is -5, as shown in the figure). The vector d_2 in Fig.5.2 represents the positional embedding of the second entity "ODC". If the sentence does not contains any entity or just one entity we use different representations of the PE vector. These representations are explained in section 5.4.

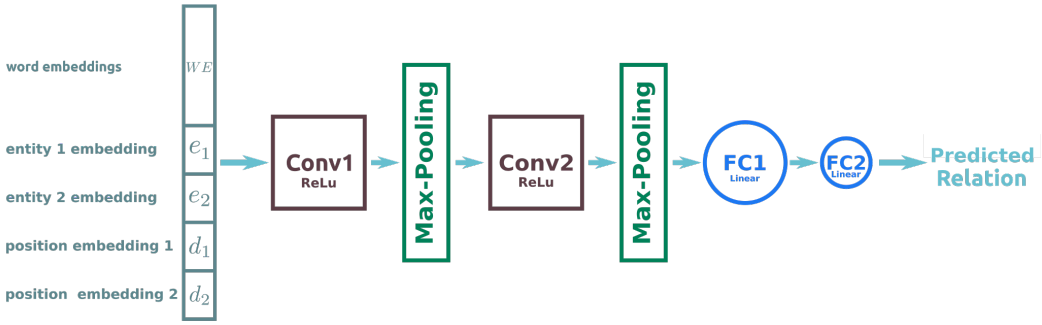

FIGURE 5.3: CNN architecture for the experiments $1-4$.

5.4 Results of CNN Model

The model used in this master thesis is based on those suggested in [[31], [11]]. In Ref.[11], the authors suggested CNN- and RNN-based models aiming to predict five possible relations (classes) between the entities pair CHEMICAL - PROTEIN. While in Ref.[31], the authors further used only CNN-based models.

In our mode, we modified the architecture by first adding a second convolutional layer and additional FC layer. The effects of the additional FC layer were minimal. The last classification FC output layer consists of 14 neurons, where the neuron with the largest activation predicts the corresponding relation. 13 out of the 14 output neurons account for the predefined relations discussed in section 5.1. The 14^{th} neuron is essential to allow the model classifying sentences where no relation exists (labeled as "no-relation", meaning either at maximum one entity occurs in the sentence or no relation exists between sentence entities).

Fig.5.3 shows the CNN architecture used for experiments. This architecture consists of the following elements:

1. Input layer: the embedding layer where the words in the sentence converted into their vector representations using BioWordVec embedding model, a pre-trained embeddings for biomedical words (with dimension equals to 200) and sentences. This embedding model is trained on PubMed articles and clinical notes from MIMIC-III Clinical Database [32], and is shown as the first layer at the lift of Fig.5.3.

2. A first CNN layer with *ReLu* activation followed by max pooling layer, as shown in the Fig.5.3 .

3. A second CNN layer with *ReLu* activation followed by max pooling layer, as shown in the Fig.5.3. As discussed in chapter 1, these previous two layers allow the model to recognize local features in the sentences. The size of the filters used in our model is 3×3.

4. The output of the max pooling layer is then directed to two subsequent FC classifications layers. As mentioned above, the last FC linear layer,

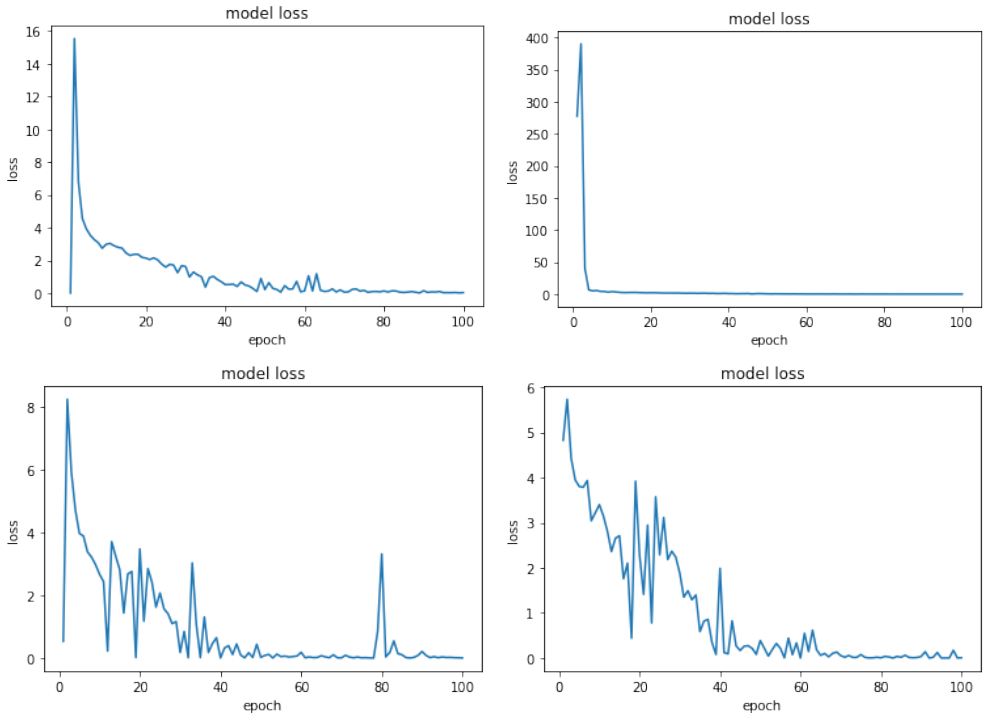

FIGURE 5.4: Loss function (from left to right) for different experiments. Experiment 1 in the top-left frame, experiment 2 (top-right), experiment 3 (bottom-left), and experiment 4 (bottom-right) -see table 5.2.

composed of 14 neurons, allows predicting the different relations that might exist in the input sequence.

The learning algorithm for this model is the stochastic back propagation with Adam optimizer, where the size of each batch is 128. The cost function is the classical *cross-entropy* loss function.

In our model, each sentence is represented by the concatenating tokens embedding, position embedding and the embedding of the two entities, then the size of the inputs will be 200 (coming from the embedding size) $\times$ 300 (coming from sentence length).

On the one hand, for the positional embeddings for the input vectors, we implemented the following choices:

- When an entity does not exist in the sentence, the positional embedding vector (d_1 or/and d_2 in Fig.5.3), assigned to this entity, is given components of incremental form, starting from 1 to the length of the sentence ("inc" in table 5.2).

- When an entity does not exist in the sentence, the positional embedding vector (d_1 or/and d_2 in Fig.5.3), assigned to this entity, is given components of decremental form, starting from $-l$, where l is the length of the sentence, to -1 ("dec" in table 5.2).

On the other hand, for the sentence padding, we first unify the lengths of different vectors in the embedding layer (PE_P), shown in Fig.5.3, by giving them the length of the word embeddings (the maximum size in the input layer). After that, we fix the size of the input layer using the length to be equal to the length of the longest sentence in the training dataset. We chose the maximum size of the input sentence to be 296. Therefore, the size of any input now is 300×200, where the additional four lines correspond to e_1, e_2, d_1, and d_2 shown in Fig.5.3. Then, we have two further choices:

- Padding the input matrix with vectors of large number (10000), given by "num_high" in table 5.2.
- Padding the input matrix with vectors of constant number (sentence length $+1$), given by "const" in table 5.2.

We performed 4 experiments using CNN-based models to predict relations between the entities CHEMICAL-GENE. Table 5.2 shows the details of these 4 experiments. The main difference between them is in the way the inputs are represented. The two main standards used here are the positional embeddings where no entity exists (PE_NE -see section 5.3) and the sentence padding (SP), discussed above.

The model has been trained for 100 epochs for all experiments. Fig.5.4 shows the evolution of loss function for the four experiments. As shown in the figure, the loss functions converge rapidly and smoothly toward small values in experiments 1 and 2 (experiment 1 on the top-left frame and experiment 2 on the top-right frame) compared to experiments 3 and 4 shown in the bottom-left and bottom-right frames of the figure, respectively. We expect that the highly osculating convergence of both experiments 3 and 4 to be a result of the sentence padding: experiments 3 and 4 use "const" padding, while experiments 1 and 2 "high_num" padding.

Table 5.2 shows the results of the 4 experiments in which we change both SP and PE_NE. As shown in the table, the highest accuracy and precision are attained by experiment 4, where we use "dec" and "const" for PE_NE and SP, respectively. This can be attributed to the relatively larger accuracy in predicting "no-relation" class, as shown in Fig.5.8.

Figs[5.5-5.8] show the confusion matrices for the four experiments performed using CNN model. As shown in these figures, their accuracy are strongly affected by the size of each relation class. However, the experiment 4, with "dec" and "const" parameters, proved to be better at predicting the "no-relation" class. It is worth mentioning that the different accuracies, reached for each class prediction, are equally treated when estimating the overall accuracy of each model (the total accuracy equals to the average of the diagonal elements of the confusion matrix).

	Acc	**P**	**R**	**F**	**PE_NE**	**SP**	**PE_P**
Exp1	66.756 %	0.699	**0.664**	**0.670**	inc	high num	sent_len+1
Exp2	66.969 %	0.678	0.590	0.611	dec	high num	sent_len+1
Exp3	66.061 %	0.678	0.650	0.644	inc	const	sent_len+1
Exp4	**67.666 %**	**0.710**	0.630	0.649	dec	const	sent_len+1

TABLE 5.2: Results of CNN based models over our test set 3748 sentences, where Acc, P, R, F, PE_NE, SP, and PE_P are the accuracy, the precision, the recall, the F1 score, the positional embeddings when no entity exists, the sentence padding, and positional embedding padding, respectively.

However, the sizes of different classes vary considerably. Therefore, the weighted accuracy of each model is larger than those seen in table 5.2. This weighted accuracy is given by

$$acc_w = \sum \frac{\omega_i \mathcal{C}_{ii}}{S_C} \tag{5.2}$$

where acc_w is the weighted accuracy, ω_i is the weight of the diagonal element $\mathcal{C}_{ii}$, and S_C is the size of the confusion matrix.

Therefore, in order to obtain a better model in our example, we can balance the classes using oversampling techniques or group minority relations together. Another solution to improve the performance is to increase the number of training samples for minority classes. In fact, one can easily recognise, by looking at Figs[5.5-5.8], that the accuracies of minority classes are very disperse, which as mentioned before leads to lower accuracy of the model.

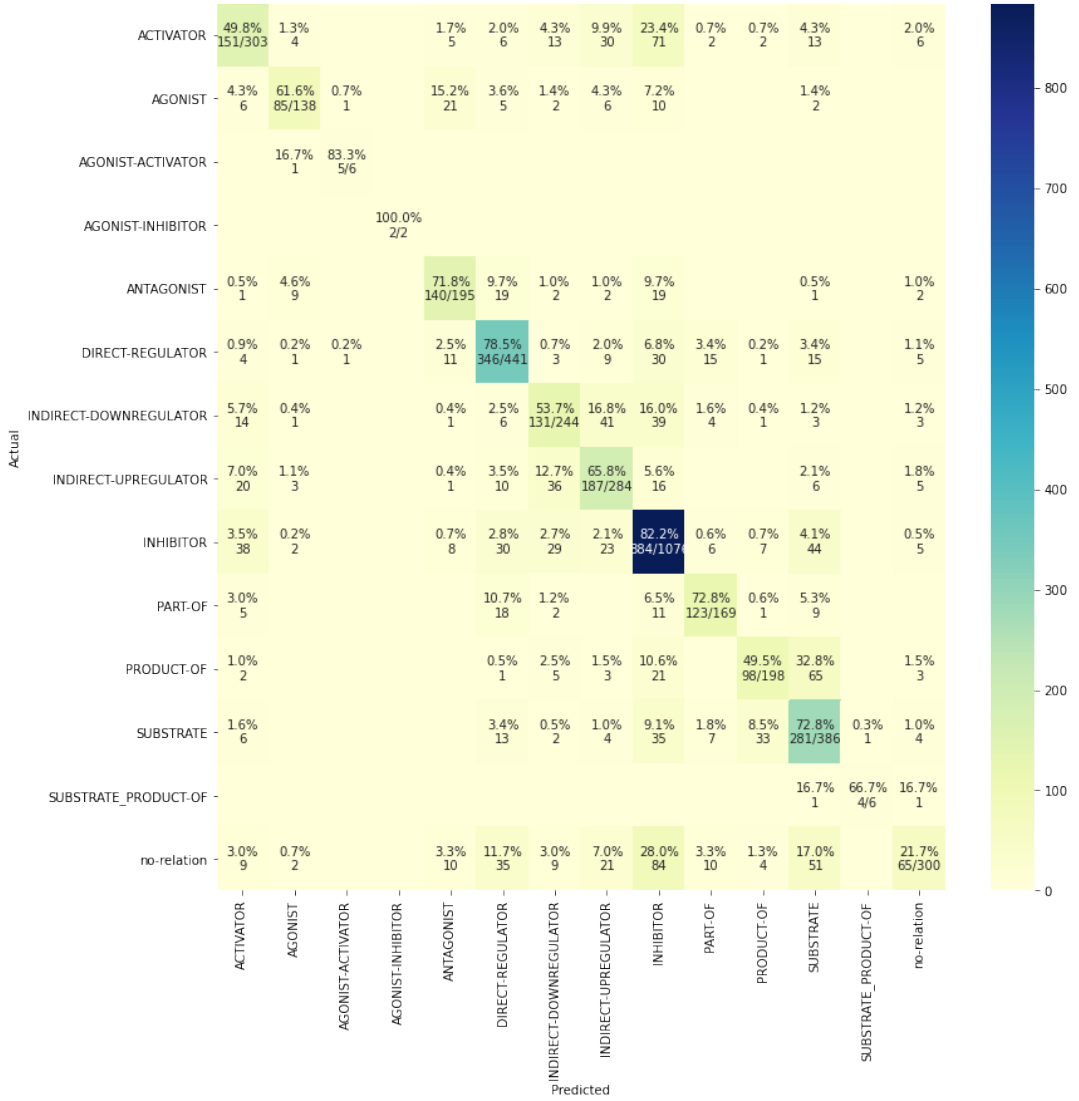

FIGURE 5.5: Confusion matrix of experiment 1 using CNN model of table 5.2.

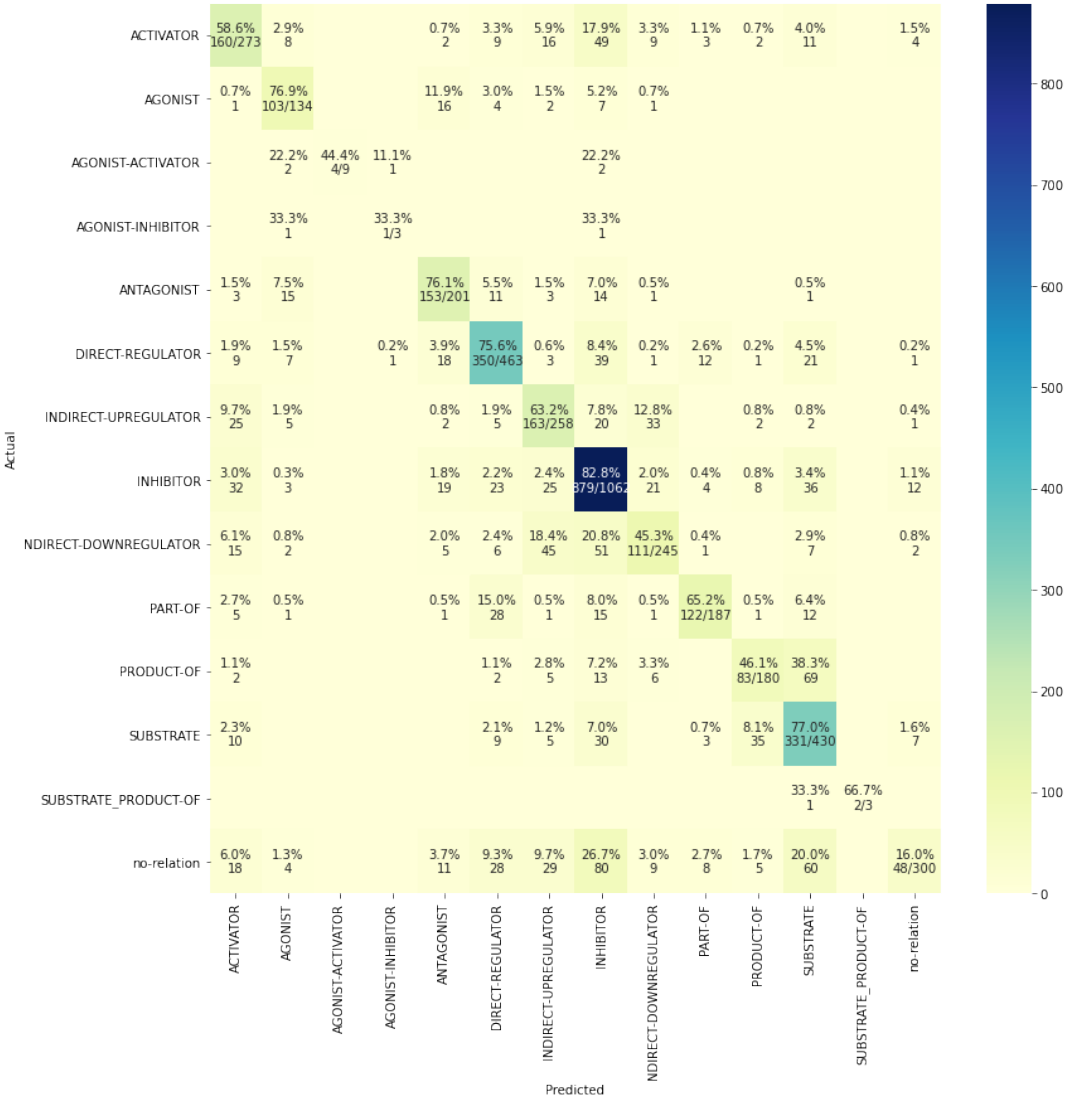

FIGURE 5.6: Confusion matrix of experiment 2 using CNN model of table 5.2.

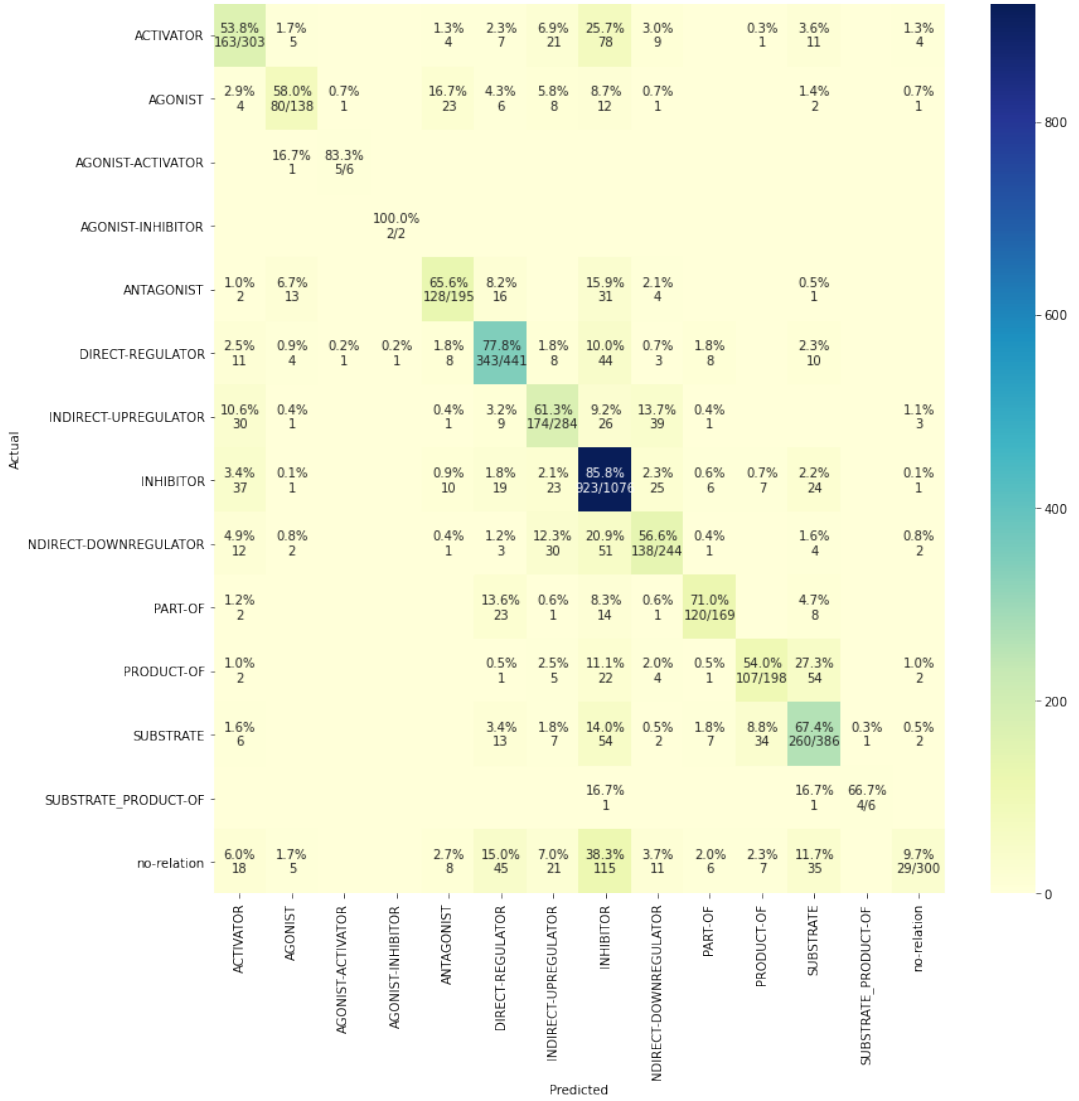

FIGURE 5.7: Confusion matrix of experiment 3 using CNN model of table 5.2.

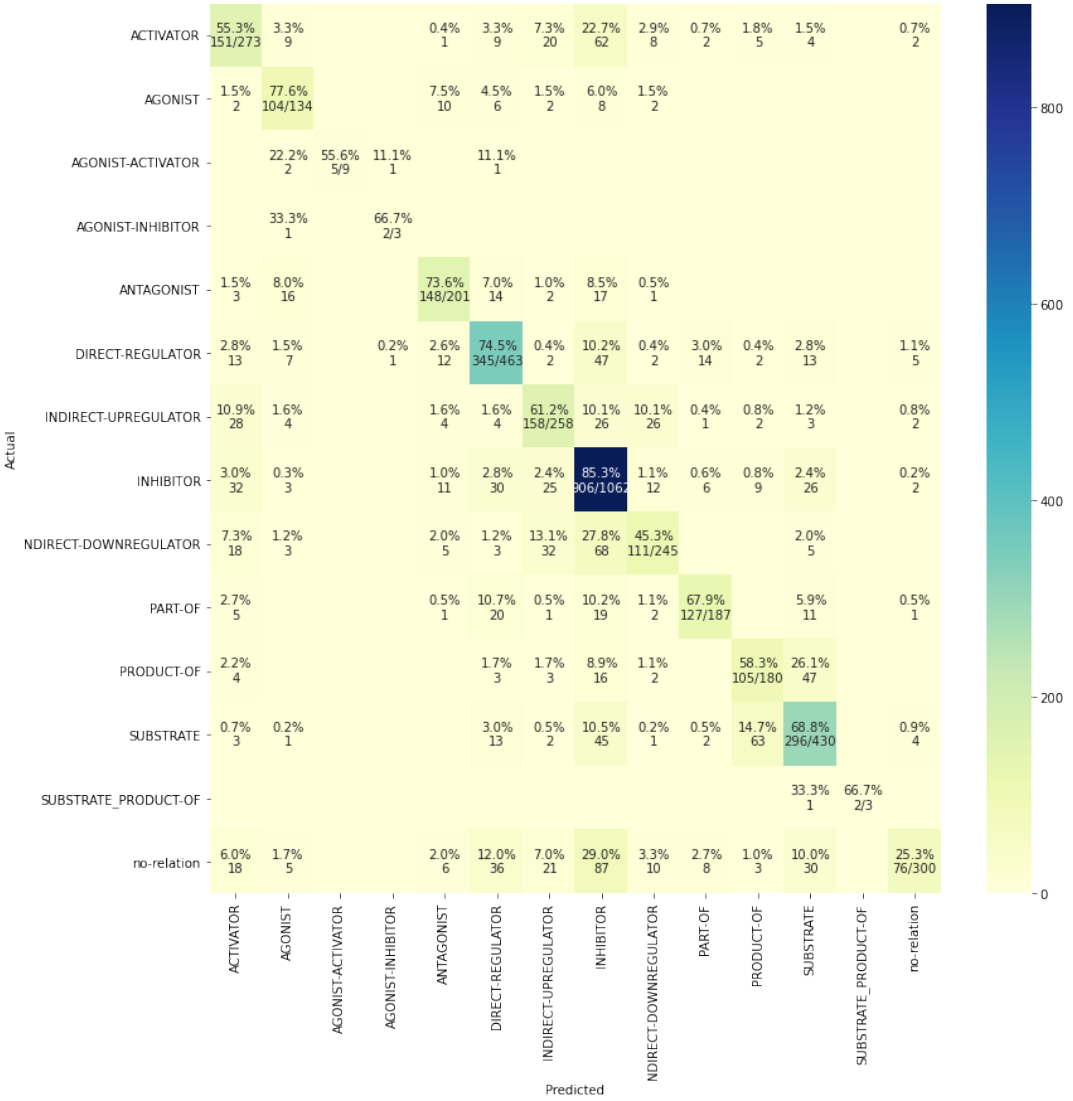

FIGURE 5.8: Confusion matrix of experiment 4 using CNN model of table 5.2.

5.5 Results of SciBERT and Hybrid Models

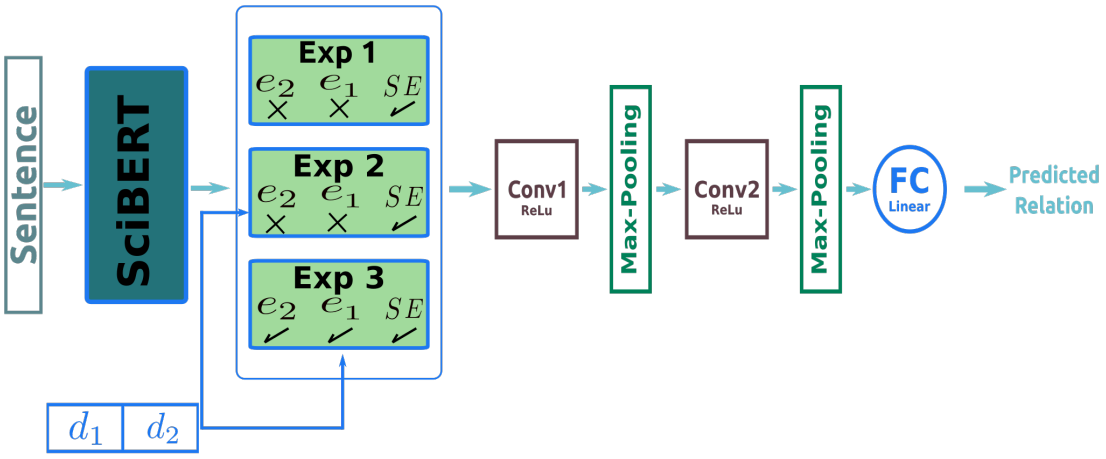

FIGURE 5.9: The configurations of different experiments shown in table 5.3. d_1 and d_2 stand for the positional embeddings of the corresponding entities e_1 and e_2, respectively, as detailed in section 5.3 and graphically shown in Fig.5.2.

In addition to using models relying only on CNNs as discussed in the previous section, we also used SciBERT model to obtain the representation of the input sentences. For all experiments considered in this section, SciBERT model represents the input layer which provides the sentence embeddings. The main configurations of the experiments which use SciBERT model (see Fig.5.9 and table 5.3) are:

1. Experiment 1 ("Exp1" in table 5.3). Here we use SciBERT model only for sentence embeddings. In this case, we do not consider entities and positional embeddings, as shown in Fig.5.9.

2. Experiment 2 uses only the positional embeddings, while SciBERT model provides only the sentence embeddings.

3. Experiment 3 and 4: these experiments use positional embeddings and SciBERT model gives both sentence and entities embeddings, as shown in "Exp3" block in Fig.5.9.

The output of SciBERT is then passed to two subsequent blocks. Each block consists of a convolutional sub-layer (shown as brown square in Fig.5.9) followed by a max pooling one to obtain a compact representation of the output of each convolutional layer (shown as green rectangle in Fig.5.9).
In experiment 3 and 4 explained above, we applied the early stop technique to avoid over-fitting, where we stop training after 7 epochs if no improvement on the test set accuracy achieved.

Fig.5.10 shows the accuracies evolution of experiments 3 and 4 of table 5.3. We notice that we reach the same accuracy in experiments with a smaller

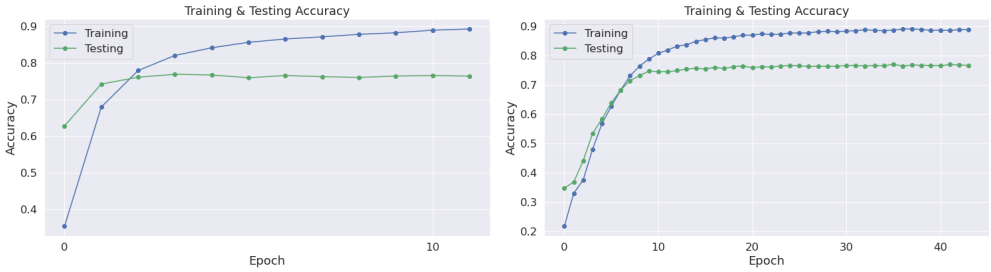

FIGURE 5.10: The accuracies of experiments 3 and 4, shown in table 5.3, at right and left frames respectively.

number of epochs, this might be understood as a consequence of a smaller size of the batches used in experiment 4 compared to experiment 3.

We also notice a slight improvement in the performance of the model when entity embeddings are applied, as can be seen when comparing experiments 3 and 2 (see table 5.3). This might be a direct consequence of the certainty imposed on the two entities whose relation is to be predicted. This is not true when comparing experiment 1 with other experiments. In this case, we observed about 7% improvement in the accuracies of experiments $2-4$ compared to experiment 1. Therefore, we expect this improvement to be a result of positional embeddings.

Figs.[5.11 - 5.13] show the confusion matrices for the experiments $1-3$, when comparing these figures, we notice that the influence of positional embeddings on predicting the "no-relation" class is significant. The accuracies of predicting the "no-relation" class are 73% and 70% for experiments 2 and 3, respectively. However, it is much smaller for experiment 1: about 38%.

Comparing the results of the model based only on CNN architecture (see section 5.4) and the hybrid models discussed in this section, two main improvements are obvious. The first is about 10% increase in the accuracy and precision is observed in hybrid models. The second interesting observation is the ability of hybrid models to reach their high accuracies on the same corpus (DrugProt) with much less number of epochs. In fact, this due to the language model of the input obtained thanks to SciBERT layer, as shown in Fig.5.9. As expected, the representations provided by SciBERT model, presented in details in chapter 4, contain much contextual information about the sentences compared to embeddings methods for CNN based model (BioWordVec embedding method).

	Acc	P	R	F1	PE	EE	EP	B-S
Exp1	70%	0.63	0.58	0.60	-	-	50	32
Exp2	76%	0.66	0.59	0.61	+	-	60	32
Exp3	77%	0.66	0.60	0.62	+	+	44	64
Exp4	77%	0.60	0.57	0.59	+	+	12	32

TABLE 5.3: Results of the experiments based on hybrid models on our test set, where Acc, P, R, F, PE, EE, EP, B-S are the accuracy, precision, recall, F1 score, positional embeddings, entity embeddings, epochs, and batch size, respectively.

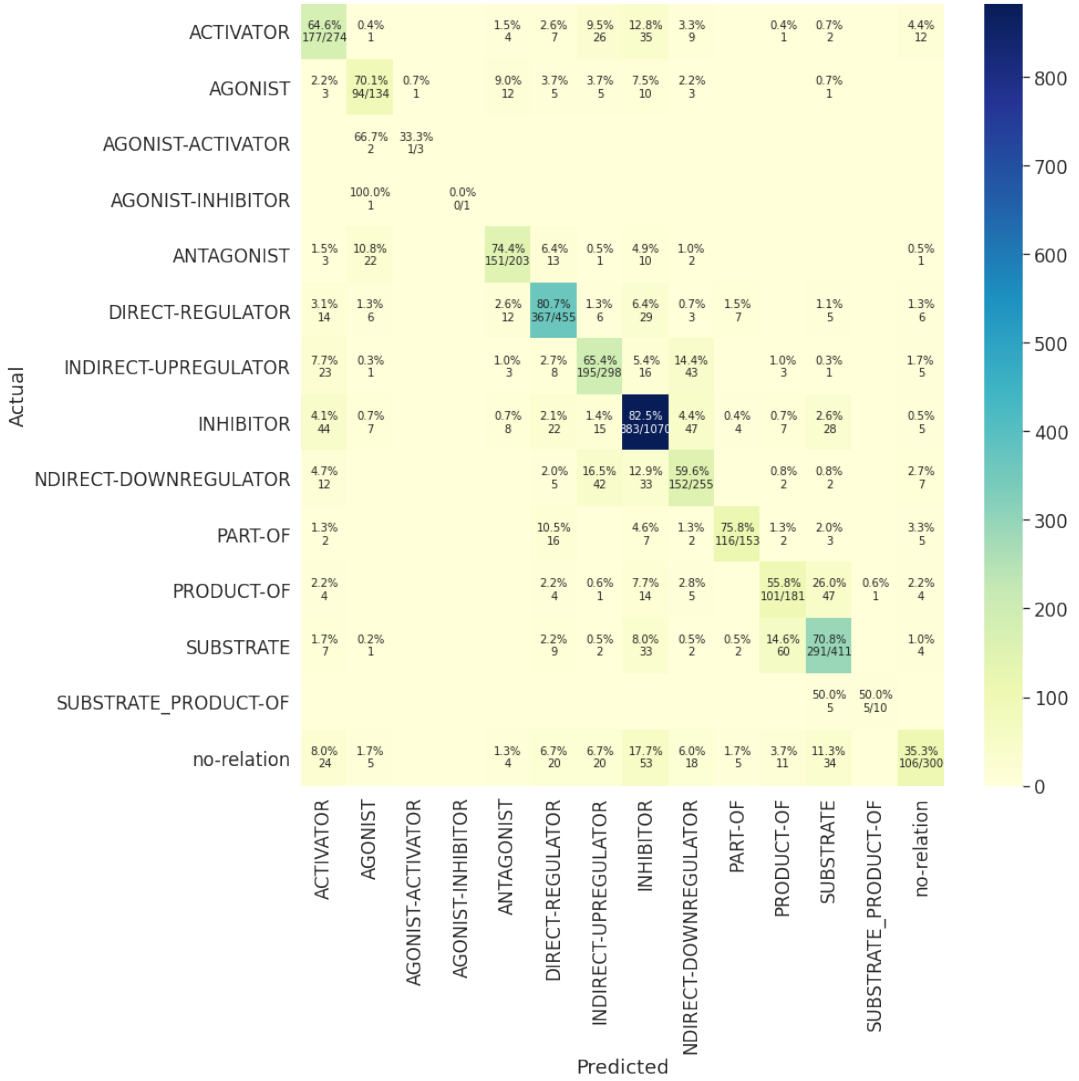

FIGURE 5.11: Confusion matrix of SciBERT model without positional embedding ("Exp1" in table 5.3).

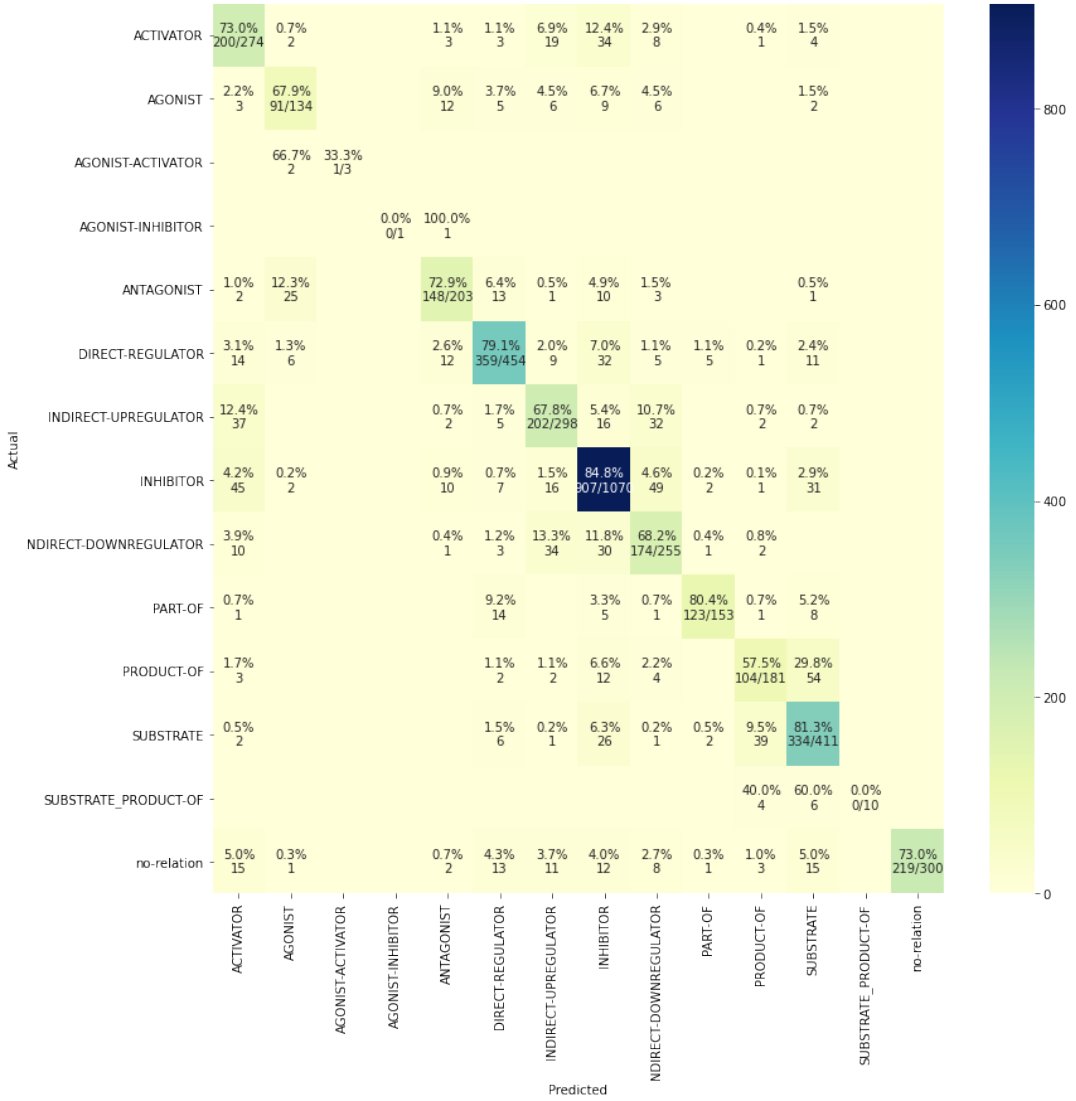

FIGURE 5.12: Confusion matrix of the hybrid model (SciBERT + CNN) with positional embedding ("Exp2" in table 5.3).

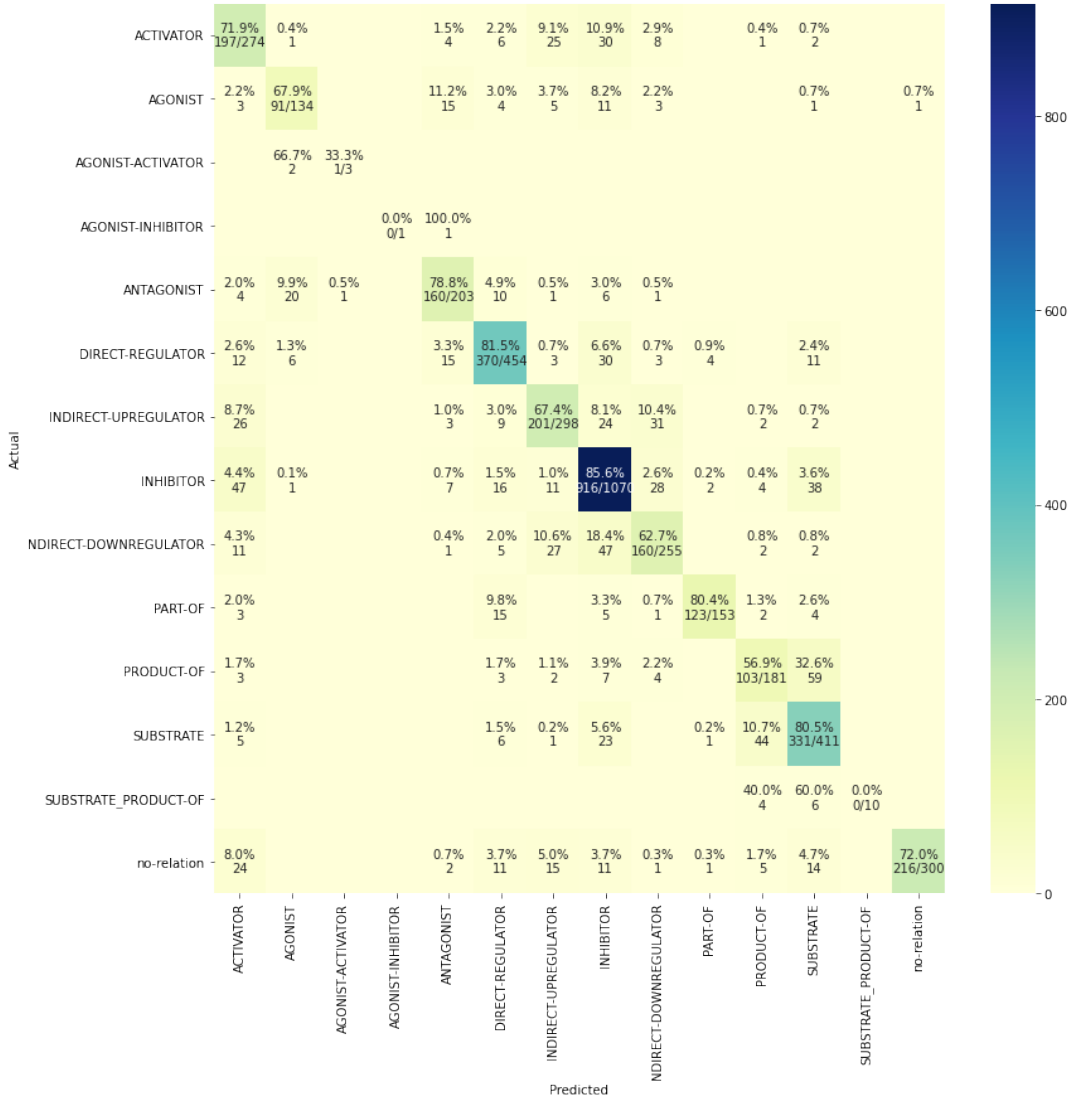

FIGURE 5.13: Confusion matrix the hybrid model (SciBERT + CNN) with positional and entity embeddings ("Exp3" in table 5.3).

Chapter 6

Conclusions and Future Work

BERT model [19] is a state of art language model based on the transformer encoder [20] which gives an intermediate highly contextualized representations.

In this master thesis, we present preliminary results of using CNN-based models and hybrid (BERT + CNN) models for relation extraction tasks on DrugProt corpus. This corpus was created by BioCreative organization that proposed several NLP challenges in biomedical domain. We participate now in one of these challenges: *BioCreative VII-Track1*shared task.

In this work, we observed that the positional embeddings play a key role in enhancing the performance of both models. However, the BERT-based models outperform those of CNN by approximately 10% in accuracy. These more accurate results obtained by BERT models have been also reached in a smaller number of epochs compared to those in CNN models, which can be attributed to the rich contextualized representations, given by BERT.

By looking at the results, we also observe that we have to treat unbalanced data. For example, for the "INHIBITOR" relation, there are 1149 samples. At the same time, there exist other relations with numbers of samples less than 30, such as the "AGONIST-INHIBITOR" relation which has only 13 samples. To overcome this problem, we suggest to apply oversampling techniques and to use external resources.

Another interesting suggestion is to add an LSTM layer. This layer can be added either before the output layer or directly after the BERT one in hybrid-based models. This addition might help capturing more context from LSTM input comings from the previous layer.

Since our results are preliminary, we are going to perform additional set of experiments that improve the comparison between different models and allow us to obtain a deeper understanding for different parameters that can have significant effects on the models performance. This further experiments are also important for our participation in the *BioCreative VII-Track1* shared task.

Bibliography

[1] M. A. Nielsen, *Neural networks and deep learning*, misc, 2018. [Online]. Available: http://neuralnetworksanddeeplearning.com/.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[3] F. Chollet, *Deep Learning with Python*. Manning, Nov. 2017.

[4] A. LeNail, *Nn-svg*, http://alexlenail.me/NN-SVG/index.html, 2021.

[5] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[6] K. D. Foote, *A brief history of deep learning*, Feb. 2017.

[7] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 2004.

[8] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. DOI: 10.1162/neco.1989.1.4.541.

[9] BioCreative, *Biocreative*, 2004.

[10] M. Krallinger, O. Rabal, S. Akhondi, M. Pérez, J. Santamaría, G. P. Rodríguez, G. Tsatsaronis, A. Intxaurrondo, J. A. B. López, U. Nandal, E. V. Buel, A. Chandrasekhar, M. Rodenburg, A. Lægreid, M. A. Doornenbal, J. Oyarzábal, A. Lourenço, and A. Valencia, "Overview of the biocreative vi chemical-protein interaction track," 2017.

[11] Y. Peng, A. Rios, R. Kavuluru, and Z. Lu, "Chemical-protein relation extraction with ensembles of svm, cnn, and RNN models," *CoRR*, vol. abs / 1802.01255, 2018.

[12] S. Matos, "Extracting chemical–protein interactions using long short-term memory networks," in *Proceedings of the BioCreative VI Workshop*, 2017, pp. 151–154.

[13] D. Jurafsky and J. H. Martin, "Information extraction," in *Speech and Language Processing*, Stanford University, 2020, ch. 17, pp. 332–354.

[14] J. Lee, S. Seo, and Y. S. Choi, "Semantic relation classification via bidirectional LSTM networks with entity-aware attention using latent entity typing," *CoRR*, vol. abs/1901.08163, 2019. [Online]. Available: http://arxiv.org/abs/1901.08163.

[15] N. Bach and S. Badaskar, "A review of relation extraction," 2007.

[16] D. Zhang and D. Wang, "Relation classification via recurrent neural network," *CoRR*, vol. abs/1508.01006, 2015. [Online]. Available: http://arxiv.org/abs/1508.01006.

[17] D. Zeng, K. Liu, S. Lai, G. Zhou, and J. Zhao, "Relation classification via convolutional deep neural network," in *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, Dublin, Ireland: Dublin City University and Association for Computational Linguistics, Aug. 2014.

[18] T. Nguyen and R. Grishman, "Relation extraction: Perspective from convolutional neural networks," in *VS@HLT-NAACL*, 2015.

[19] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. arXiv: 1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, 6000–6010, ISBN: 9781510860964.

[21] A. Rajaraman and J. D. Ullman, "Data mining," in *Mining of Massive Datasets*. Cambridge University Press, 2011, 1–17.

[22] G. Boleda, "Distributional semantics and linguistic theory," *Annual Review of Linguistics*, vol. 6, pp. 213–234, 2020.

[23] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. [Online]. Available: https://aclanthology.org/N18-1202.

[24] B. colah, *Understanding lstm networks*, Aug. 2015.

[25] J. Alammar, *The illustrated bert, elmo, and co. (how nlp cracked transfer learning)*, https://jalammar.github.io/illustrated-bert/, 2021.

[26] I. Beltagy, A. Cohan, and K. Lo, "Scibert: Pretrained contextualized embeddings for scientific text," *CoRR*, vol. abs/1903.10676, 2019. arXiv: 1903.10676. [Online]. Available: http://arxiv.org/abs/1903.10676.

[27] W. Hafiane, J. Legrand, Y. Toussaint, and A. Coulet, "Experiments on transfer learning architectures for biomedical relation extraction," *CoRR*, vol. abs/2011.12380, 2020. [Online]. Available: https://arxiv.org/abs/2011.12380.

[28] J. Kringelum, S. K. Kjaerulff, S. Brunak, O. Lund, T. I. Oprea, and O. Taboureau, "ChemProt-3.0: a global chemical biology diseases mapping," *Database*, vol. 2016, Feb. 2016, ISSN: 1758-0463.

[29] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science, I. I. Ivanov, M. van Sinderen, F. Leymann, and T. Shan, Eds., vol. 367, Springer International Publishing, 2013, pp. 3–20, ISBN: 978-3-319-04518-4. DOI: 10.1007/978-3-319-04519-1_1.

[30] M. Krallinger, O. Rabal, S. A. Akhondi, M. P. Pérez, J. Santamaría, G. P. Rodríguez, *et al.*, "Overview of the biocreative vi chemical-protein interaction track," in *Proceedings of the sixth BioCreative challenge evaluation workshop*, vol. 1, 2017, pp. 141–146.

[31] Y. Peng and Z. lu, "Deep learning for extracting protein-protein interactions from biomedical literature," Jan. 2017, pp. 29–38. DOI: 10.18653/v1/W17-2304.

[32] A. E. Johnson, T. J. Pollard, L. Shen, H. L. Li-Wei, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark, "Mimic-iii, a freely accessible critical care database," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.