h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Darmstadt University of Applied Sciences

Faculty of Mathematics and Natural Sciences & Computer Science

Language Modeling of Physics and Computer Science Texts with BERT Models

Submitted in partial fulfillment of the requirements for the degree of

Master of Science (M.Sc.)

by

Jeremy Mah Zhee Kein

Matriculation number: 769452

First Examiner	:	Prof. Dr. Stefan Rapp
Second Examiner	:	Prof. Dr. Antje Jahn

Date of Issue	:	19. January 2023
Date of Submission	:	05. July 2023

Jeremy Mah Zhee Kein: *Language Modeling of Physics and Computer Science Texts with BERT Models,* © 05. July 2023

DECLARATION

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, 05. July 2023

Jeremy Mah Zhee Kein

ABSTRACT

Recent developments in pre-trained language models advanced the field of natural language processing (NLP). The introduction of Bidirectional Encoders for Transformers (BERT) has had important impact and increased the relevance of pre-trained models. At the beginning, research in this field was started on English text data. This was followed by multilingual text corpora. Despite of this, there has been a lack of domain-specific language models, that could extract information from texts which belong to a specific field. Some examples of such language models are SciBERT, BioBERT and MatSciBERT. The use of these language models however yields subopotimal results when used in the physics and computer science domain. This work presents new BERT language models, that are pre-trained with text data from the physics and computer science domain obtained from the open-access repository arXiv. These models are then evaluated based on their performance in named entity recognition (NER) as a downstream task. We show that the models pre-trained in this work achieved lower pseudo-perplexities than their original counterparts. Additionally, we show that the models pre-trained in this work improved the micro F1 scores of the original models on the computer science and physics named entity recognition datasets by up to 0.69% and 3.85%, respectively. The addition of a Conditional Random Fields (CRF) layer however did not improve the performances of the models on the named entity recognition tasks in this work. However, the models pre-trained in this work still achieved higher micro F1 scores compared to their original counterparts regardless whether the CRF layer was used.

ZUSAMMENFASSUNG

Vor Kurzem haben vortrainierte Sprachmodelle den Bereich der Verarbeitung von natürlichen Sprachen (NLP) vorangebracht. Die Einführung von Bidirectional Encoders for Transformers (BERT) hat die Bedeutung von vortrainierten Modellen erhöht. Anfangs wurde im Bereich englischer Textdaten geforscht, gefolgt von Modellen, die mit mehrsprachigen Textkorpora trainiert wurden. Allerdings existieren inzwischen wenige domänenspezifische Sprachmodelle, die sich mit der Semantik und Syntax von Texten eines bestimmten Bereichs befassen. Beispiele für BERT-basierte domänenspezifische Sprachmodelle sind SciBERT, BioBERT und MatSciBERT. Die direkte Anwendung dieser Modelle im Bereich der Physik und der Informatik kann zu suboptimalen Ergebnissen führen, denn die Modelle sind nicht mit den für den Bereich spezifischen Bezeichnungen und Fachausdrücke trainiert. In dieser Arbeit werden neue BERT Sprachmodelle vorgestellt, die mit Hilfe von Texten aus dem Bereich der Physik und Informatik aus dem arXiv-Repository vortrainiert werden. Die neue Sprachmodelle werden anhand ihrer Leistung bei einer nachgelagerten Named-Entity-Recognition (NER) Aufgabe in den Informatik- und Physikdomänen bewertet. Die in dieser Arbeit an unserem Textkorpus vortrainierten Sprachmodelle haben kleinere Pseudo-Perplexitätswerte erreicht. Außerdem sind die Sprachmodelle auch in der Lage, höhere Mikro F1-Werte im Vergleich zu BERT und SciBERT zu erzielen. Eine Verbesserung von bis zu 0,69% und 3,89% wurden jeweils für die NER-Datensätze aus dem Informatik- bzw. Physikbereiche gemessen. Ein zusätzlicher Conditional Random Fields (CRF)-Schicht hat allerdings keine Verbesserungen zu der NER-Leistung der Modelle gebracht.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

BERT	Bidirectional Encoder Representations from Transformers
CoVe	Contextualized Word Vectors
CRF	Conditional Random Fields
CS-NER	Computer Science Named Entity Recognition in the Open Research Knowledge Graph
DEAL	Detecting Entities in the Astrophysics Literature
ELMo	Embeddings from Language Models
FFNN	Feed Forward Neural Network
GloVe	Global Vectors
LSTM	Long Short-Term Memory
MLM	Masked Language Modeling
NCG	NLPContributionGraph
NER	Named-Entity Recognition
NLP	Natural Language Processing
NSP	Next Sentence Prediction
POS	Parts-of-Speech
PwC	PapersWithCode
RNN	Recurrent Neural Network
WIESP	Workshop on Information Extraction from Scientific Publications

Part I

THESIS

1 INTRODUCTION

1.1 MOTIVATION

The abundance of computer science and physics papers have continued to rise rapidly. In 2022 , more than 13000 articles [71] in the physics sector and more than 70000 research papers in the computer science sector [70] have been published on arXiv, an open access repository for electronic preprints in the scientific field. Figure 1.1 shows the number of articles, cross listed articles not included, that have been published on arXiv annually from 2011 to 2022. With the increase in the numbers of the documents comes a demand for mining information from these text data. Though most machine learning technologies focus on structured information, there is also an abundance of information, that can be extracted from textual data. However, text data is high-dimensional, unstructured and, at times, sparse in information.

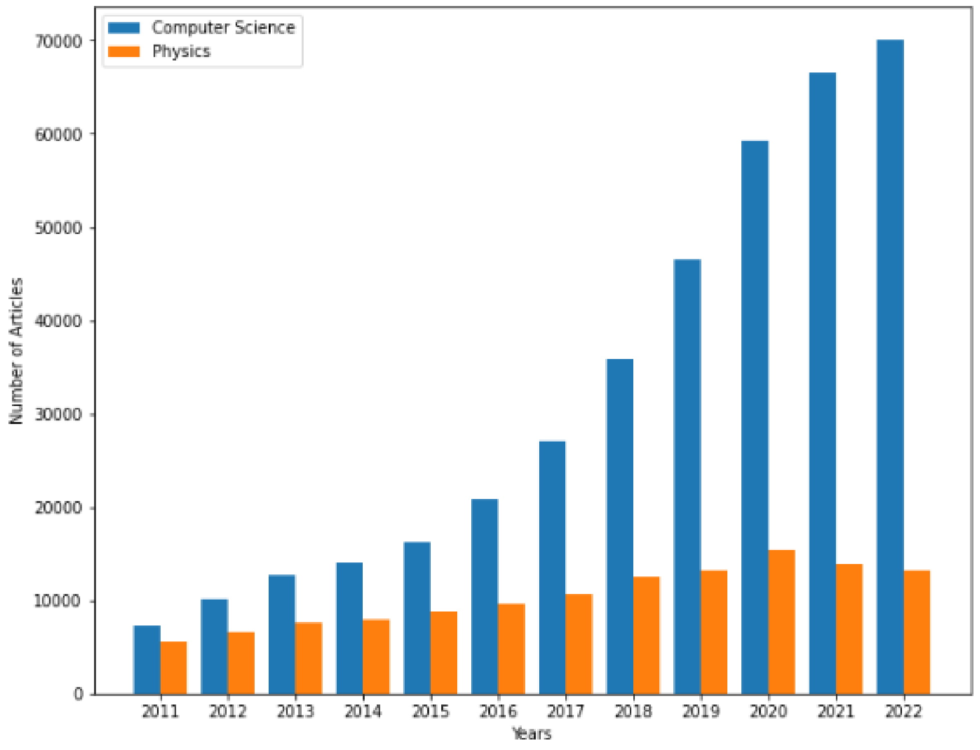


Figure 1.1: Number of articles from physics and computer science domains from arXiv [70, 71].

In the recent years, progress of text mining models has made its stride with the advancements of deep learning technologies used in the field of Natural Language Processing (NLP). To further elaborate, NLP is the application of computational methodologies for the analysis and synthesis of natural language and speech and it is therefore a multidisciplinary subfield of linguistics, artificial intelligence and computer science. For instance, the introduction of RNN [35, 67] and LSTM [17, 56, 64] have made it possible for different applications of NLP such as text generation [51] and machine translation [8]. The information found in recent text mining techniques have been used to build knowledge graphs and help in the navigation of semantic webs, in order to gain better insight and an overview of text data. However, the use of RNNs and LSTMs does come with its limitations. The former suffers from the phenomenon known as vanishing gradients, as long series of multiplications of small values diminish the gradients and cause the gradients to be insignificant, as the layers get deeper [56]. The latter is an improvement of the RNN, whereby the LSTM can retain information in memory for long periods of time and hence overcome the shortcoming of vanishing gradients in an RNN [56]. Both RNN and LSTM are nonetheless recurrent model and therefore process data sequentially, which prohibits parallelization during training. This becomes critical when longer sequence data is required for training these networks. Recent word representation models such as Word2Vec [37] and ELMo [45], which capture the syntax and semantics of a word in isolation, do not distinguish their representations or embeddings in different contexts. With the introduction of the transformer network and its derivative BERT, contextual embeddings of words can be used to model text data, that contain long-range information.

BERT was however pre-trained on datasets containing general domain texts in the English language [13]. It is hence questionable whether its performance is optimal for domain-specific tasks. Therefore, domain-specific BERT-variants have been researched in the recent years. For instance, **SciBERT**, is one of the very first variants of BERT that was pre-trained on unlabeled scientific texts from biomedical and computer science literature, in order to model texts in the scientific domain [4]. BioBERT was pre-trained with BERT on biomedical text corpora [28]. ClinicalBERT was similarly pre-trained with BERT on clinical notes to predict hospital readmission based on clinical notes [18]. To the best of our knowledge, no BERT variants for the modelling of a combination of texts from the computer science and physics domain have been studied.

1.2 GOAL OF THIS THESIS

Since computer science plays an important role in the physics domain, it would be interesting to pre-train a BERT model in both physics and computer science domains. Although **SciBERT** was also pre-trained on computer science texts, it is interesting to investigate, whether further expanding the unlabeled text corpus for pre-training **SciBERT** on computer science texts will improve its performance on downstream tasks in these domains. Hence, the goal of this thesis is to model texts from the physics and computer science domain using a method known as masked language modelling with the help of BERT models initiated with weights from **SciBERT** and **BERT$_{\mathbf{BASE}}$**. **BERT$_{\mathbf{BASE}}$** was the first BERT model pre-trained on English texts from Wikipedia and BooksCorpus [13]. The models pre-trained on texts from computer science and physics domains and initialized with **BERT$_{\mathbf{BASE}}$** and **SciBERT** weights in this thesis shall henceforth be known as PCBERT and PCSciBERT, respectively. PCBERT denotes "**P**hysics **C**omputer Science BERT" and PCSciBERT denotes "**P**hysics **C**omputer Science **SciBERT**". Both uncased and cased variants of the models were used in this thesis to study the effects of case sensitivity on their ability to model the language used in these two domains, mainly because it is expected that case-sensitivity is important for our downstream task. The texts that are used for the pre-training stage are collected and extracted from the open access repository arXiv.

In order to evaluate the performance of the models, the models are fine-tuned for domain-specific Named-Entity Recognition (NER) tasks. The two NER datasets for physics and computer science domains originate from WIESP dataset [65] as well as the CS-NER dataset [12], respectively. The former proposes text fragments from astrophysics papers provided by NASA Astrophysical Data System with manually tagged astronomical facilities and other entities of interest. On the other hand, the latter provides a standardized dataset for NER in the computer science domain by defining a set of contribution-centric scholarly entities. A summary of the workflow for this thesis is provided in Figure 1.2. Further studies from [1, 57] have also introduced the application of CRF on language models for the improvement of its performance in sequence labeling tasks. It would therefore also be in this work's interest to investigate whether the addition of a CRF layer on the pre-trained language models would perform better than its counterparts without the said layer.

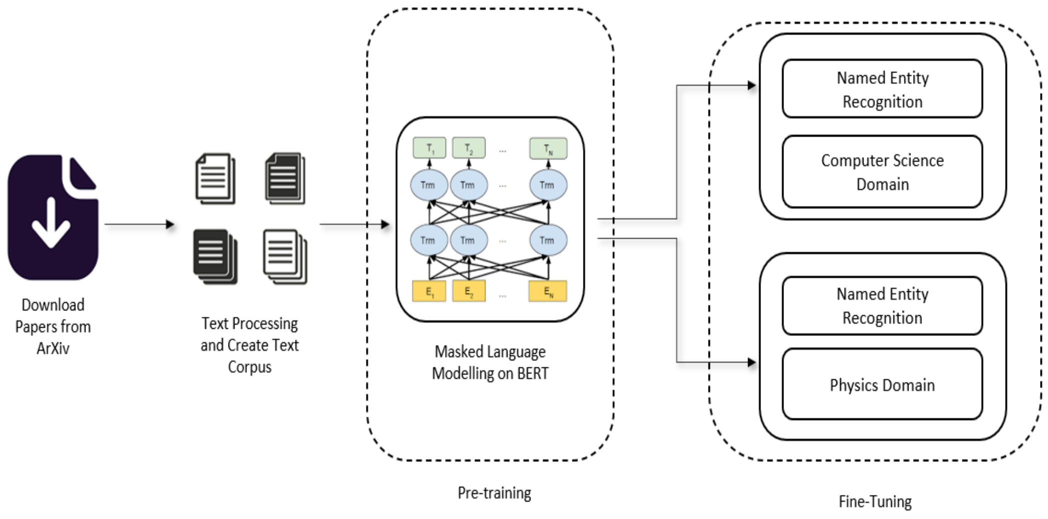


Figure 1.2: Workflow for producing PCBERT and PCSciBERT models.

1.3 THESIS STRUCTURE

In the next chapter, the theoretical background of language modelling and the BERT architecture as well as the concept of NER is introduced. In chapter 3, the process of text extraction from arXiv and further details on the CS-NER and WIESP datasets are described. Expanding upon that, the methods used for pre-training and fine-tuning are also shown. Chapter 4 shows the results obtained in this work and its related discussion. Lastly, Chapter 5 presents the summary of this work in addition to further work that may be explored upon the findings of this thesis.

2 THEORETICAL BACKGROUND

2.1 LANGUAGE MODELLING

A language model has the objective to learn the true distribution of a text corpus and hence attempts to model a probability distribution over sequences of tokens or words. Given a vocabulary V of a set of word tokens, a language model assigns each sequence of tokens $x_1, ..., x_L \in V$ a probability, where x is a token and L is the total number of tokens in a sentence [52]. For instance, a language model should assign the sentence "The fox jumped over the fence" with a high probability and the sentence "The fox jump over the fence" with a lower probability due to its grammatical error and the infrequent mention of "jump" appearing directly after "fox" in a sentence.

Perhaps the most useful method to learn the true distribution of a text corpus is through next word prediction. Given a sentence s, the goal of next word prediction is to predict a distribution of $P(x|s)$ of the following word, where $P(x|s)$ denotes the conditional probability of a word token, given the previous sequence of words [52]. For example, in the incomplete sentence "The cat chased the", the language model would place words such as "mouse", "bird" or "insect" with a high probability.

The simplest form of a language model would be the n-gram model. An n-gram is commonly defined as a sequence of n words, e.g. a 2-gram or bigram is sequence containing two words such as "cat chased" or "chased the" from the previously mentioned example and a 3-gram, also known as a trigram, contains the 3 words "cat chased the". In an n-gram model, the probability of words are calculated using relative frequencies of word counts, i.e. a large text corpus is created and the number of occurrences of the word in focus preceded by the previous words in the sentence is counted and is compared with the number of occurrences of the previous words in the sentence. The following equation formulates this mathematically, $c(.)$ in this equation denotes the count of occurrences of these words appearing together in the text corpus:

$$P(\text{mouse}|\text{The cat chased the}) = \frac{c(\text{The cat chased the mouse})}{c(\text{The cat chased the})} \tag{2.1}$$

Since a text corpus is large and words in sentences vary, finding an exact match of the words occurring in a sequence may lead to negligible counts or even no counts at all depending on the corpus. Hence, a more practical way to compute probabilities of entire sentences would be to use the chain rule of probability to decompose the entire sequence and calculate its joint probability, as shown in the equation :

$$P(x_1, ..., x_L) = P(x_1)P(x_2|x_1)P(x_3|x_{1:2})...P(x_L|x_{1:L-1}) = \prod_{i=1}^{L} P(x_i|x_{1:i-1}) \tag{2.2}$$

$P(x_1, ..., x_L)$ denotes the joint probability of an entire sequence of words, whereas $P(x_L|x_{1:L-1})$ denotes the conditional probability of the word token x_L, given the word tokens $x_1, ...x_{L-1}$ appearing before it in the sequence. However, this calculation method still requires $P(x_L|x_{1:L-1})$ to be computed, which in large corpora is difficult to calculate, as its frequency is sparse. To alleviate this, the n-gram model approximates the probability the entire sequence of words with the probability of a word occurring with the last few words in the sentence [24]. By doing so, the n-gram model utilizes the Markov assumption, that assumes the probability of a future word token can be predicted, without looking too far in the past [24]. This assumption can be, in the case of a trigram, summarized in the following approximation:

$$P(x_L, x_{1:L}) \approx P(x_L|x_{L-1}, x_{L-2}) \tag{2.3}$$

Hence, Equation 2.2 can be, in the case of a trigram, instead be approximated with:

$$P(x_L, ..., x_L) \approx \prod_{i=1}^{L} P(x_i|x_{i-1}, x_{i-2}) \tag{2.4}$$

Although this assumption eases the calculation of long sentences, this does not reflect the syntax and semantics in natural languages well. It also does not take into account long dependencies within sentences, where one word in a sentence depends on the reference of another in the same sentence.

A frequent approach to finding the optimal solution to the language modelling problem is to minimize the cross entropy loss between the true distribution $p^t_{.|s}$ and the model prediction $p_{.|s}$ for a given sentence $s \in S$ [3, 13, 52]. p_L is used to represent the true distribution over a context set S in the corpus used for the language model [52].

The cross-entropy loss for a language model can be consequently be expressed as:

$$\mathcal{L}_{CE} = \mathbb{E}_{s \sim p_L} \mathbb{E}_{x \sim p^t_{.|s}} [-log(p_{.|s}(x))] \tag{2.5}$$

2.1.1 *Neural Language Models*

In contrast to n-gram language models, the conditional probability distribution $p_{.|s}$ is parametrized with the help of low-dimensional vectors, called embeddings. Embeddings overcome the curse of dimensionality by learning a distributed representation for words, which constitutes a word as a low dimensional vector [22]. For an embedding $w \in \mathbb{R}^d$, the softmax distribution over V using word embeddings $\Phi \in \mathbb{R}^{dx|V|}$ is :

$$p_{w,\Phi}(x) = \frac{e^{w^T \phi_x}}{\sum_{x \in X} e^{w^T \phi_x}} \tag{2.6}$$

In the case of neural language models, the context or sentence s is first embedded into $f(s) \in \mathbb{R}^d$ by employing a feature map $f : S \to \mathbb{R}^d$. The feature map chosen to embed the context can be parametrized by the neural network model of choice. The model prediction $p_{.|s}(x)$ from 2.5 can be replaced by $p_{f(s)}(x)$, that is, the predicted conditional distribution is determined by the softmax distribution influenced by the word embeddings Φ and the context embedding $f(s)$ [52]. Therefore, the cross entropy loss for a neural language model is:

$$\begin{aligned} \mathcal{L}_{CE} &= \mathbb{E}_{s \sim p_L} \mathbb{E}_{x \sim p^t_{.|s}} [-log(p_{f(s)}(x))] \\ &= \mathbb{E}_{s \sim p_L} \mathbb{E}_{x \sim p^t_{.|s}} [-log(\frac{e^{f(s)^T \phi_x}}{\sum_{x \in X} e^{f(s)^T \phi_x}})] \end{aligned} \tag{2.7}$$

2.2 WORD EMBEDDINGS

After introducing the role of word embeddings in 2.1.1 in neural language models, this section discusses the two main distinct types of word embeddings , namely the static word embeddings and the contextualized embeddings. Word embedding is a method in NLP, whereby text from a corpus is mapped to a vector. That way, words with the same meaning have the same learned representation. Therefore, words can have a more expressive representation in lower dimensions and does not suffer from the curse of dimensionality. Static embeddings represent individual words as a fixed vector and in contrast, contextualized embeddings takes into account the contextual information of a sentence or paragraph. For a static embedding, the same word in a sequence would receive the same embedding from the system, regardless of whether the sentence or context affects its meaning. The same is not true for a contextualized embedding where the same word in different contexts would receive different embeddings from the model.

2.2.1 *Static Embeddings*

2.2.1.1 *Feed Forward Neural Network Language Models*

The first ever successful implementation of a neural network model for the purpose of language modelling was introduced by Bengio, Ducharme, and Vincent [5]. In their findings, they managed to build a model that simultaneously learns a distributed representation of each word as well as the probability function for word sequences, which were expressed in the form of these representations. Similar to traditional n-gram language models, Feed Forward Neural Network (FFNN) language models are able to use $n-1$ words in a sequence to predict the nth word in the same sequence. These models were then usually used in the first layer of a deep neural network model. One drawback of using such FFNNs is the limitation of not being able to directly process variable-length data and represent the historical context. Hence for language modeling tasks FFNNs have to use fixed-length sequences as inputs [22]. The FFNN language model can be expressed mathematically as:

$$y = W * w + U * tanh(d + H * w) + b \tag{2.8}$$

W, U and H are weight matrices for the connections between the layers, whereas d and b are the biases of the hidden layer. For the model to predict the conditional probability of a word at position n, the previous $n-1$ words are projected by a shared projection matrix $C \in \mathbb{R}^{|V| \times d}$ into a continuous feature vector space based on their index in the vocabulary [5]. $|V|$ implies the size of the vocabulary. The input for the FFNN model is a concatenation of feature vectors of $n-1$ words represented by w in 2.8. The output of the model is subjected to a Softmax layer to ensure that the conditional probabilities of the words are summing to one and are positive. The architecture of this model is illustrated in Figure 2.1.

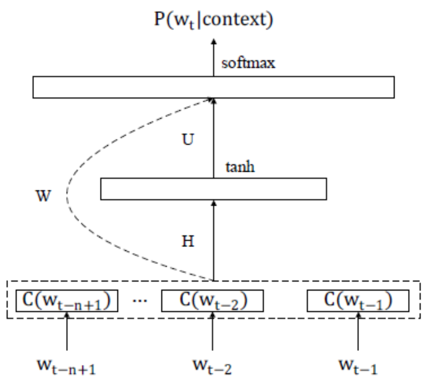


Figure 2.1: The FFNN Language Model as introduced in [5].

2.2.1.2 *Recurrent Neural Networks (RNN)*

A major drawback from Bengio's work [5] as mentioned in Subsection 2.2.1.1 is that a feedforward network has to make use of a fixed length context that needs to be defined before training. This causes the neural network to only process a fixed number of preceding words when predicting the target word. The first RNN language model was introduced in [38]. It was also suggested by the authors in [5], that utilizing more structure and parameter sharing in neural networks could capture longer contextual information.

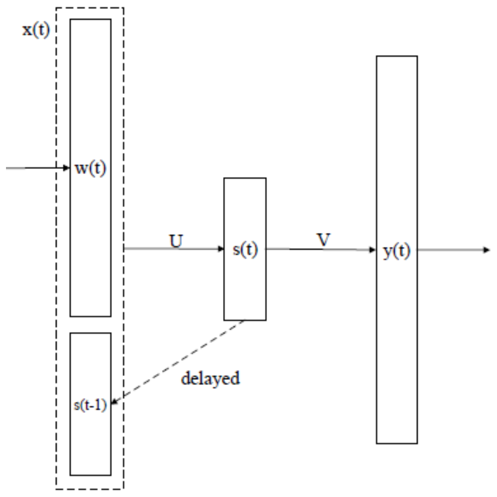


Figure 2.2: RNN shown as one layer. Each layer in the RNN takes in an input for the current time step and a state from the previous time step [22].

Figure 2.2 shows the structure of one layer in an RNN model. The model used in [38] was the simplest version of a recurrent neural network. The network comprises of an input layer x, hidden layer s which is also known as state or context layer and finally the output layer y. At a given time t, the input to the network is $x(t)$ and the state is given as $s(t)$ and finally the output is denoted as $y(t)$. Similar to an FFNN language model in Subsection 2.2.1.1, the input token is first mapped to an embedding. The input at time t is made up of the embedding $w(t)$ of the current word and the state s at time $t-1$ by concatenating both of these vectors. The structure of the RNN language model can be described as:

$$x(t) = w(t) + s(t-1) \tag{2.9}$$

$$s(t) = f(U * x(t) + b) \tag{2.10}$$

$$y(t) = g(V * s(t) + d) \tag{2.11}$$

where f(.) signifies a sigmoid activation function and g(.) is a softmax function. U and V are weight matrices , whereas b and d are biases of the state layer and the output layer correspondingly. The size of the input vector x is equal to the size of the vocabulary used during training i.e. approximately 30000 and the size of the state s is normally around 30 to 500 hidden units [38]. The RNN language model was a breakthrough approach when compared to the FFNN language model [22, 35]. Unlike FFNNs , RNNs utilize their internal memory and can process arbitrary sequences of inputs. Hence, this makes it perfect for processing sequences of data with variable lengths. The passing of internal states to neighbouring neurons at time step t also reveals timing information in the data. Though RNNs are able to take into consideration all contexts for prediction and are able to process data sequences, their major drawback can be observed when attempting to learn long-term dependencies, especially in text data. This is due to their gradient's tendency to diminish or explode as the sequence gets longer [22].

2.2.1.3 *Long Short-Term Memory (LSTM)*

The vanishing and exploding gradient problem of the RNN was studied in [17, 22]. During backpropagation through time in the RNN, the gradients are scaled by a certain factor. This effect gets magnified, as the time steps of the network increases. By introducing gate units in a network similar to RNN, the authors in [17] were successful in dropping out unimportant hidden states from previous time steps and keeping important features from the previous time steps when inputs are passed through the recurrent network. Thus, they have effectively modified the network in such a way that the scaling factor for the gradients is fixed to one and by doing so, helps the network to retain some memory when processing the input sequence. The flow of information in the LSTM is controlled by three gate structures: input gate, forget gate, and output gate. Figure 2.3 exhibits the inner structure of an LSTM network.

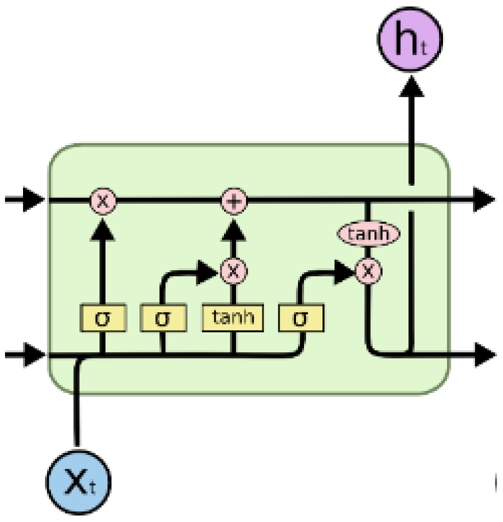


Figure 2.3: Inner workings of an LSTM network [17].

The forget gate in the network acts as a binary mask that is multiplied with the cell state of the previous time step. By applying a sigmoid function, the forget gate can decide which features from the previous cell state is kept or left out. The input gate uses two activation functions, namely the sigmoid and the hyperbolic tangent function (tanh). The tanh outputs values in$[-1, 1]$ and helps determine if a cell state should be incremented or decremented. The output gate determines the output and cell state that is exposed to the following time step of the LSTM network. According to the authors in [17], the output gate also prevents the network's attempts at storing long time lag memories from unsettling the activations that represent the learnable short time lag memories. The structure of the network can be generalized as follows:

$$
\begin{aligned}
i_t &= \sigma(W_i(h_{t-1}, x_t) + b_i), \\
f_t &= \sigma(W_f(h_{t-1}, x_t) + b_f), \\
o_t &= \sigma(W_o(h_{t-1}, x_t) + b_o), \\
\tilde{c}_t &= tanh(W_c(h_{t-1}, x_t) + b_c), \\
c_t &= f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t, \\
h_t &= o_t \cdot tanh(c_t)
\end{aligned}
\tag{2.12}
$$

where i_t, f_t, o_t are the input gate, forget gate and output gate at the current time step, respectively. h_t, h_{t-1} are the hidden states from the current and previous time step, x_t is the input at the current time step and σ is a sigmoid function. W_i, W_f, W_o are the weight matrices for the input gate, forget gate and output gate, respectively, whereas b_i, b_f, b_o are the corresponding biases. $\tilde{c}_t$ represents the candidate cell state and, which is used to calculate the current cell state c_t. The current cell state can then be outputted and be used as input in the next time step. Note that the $(\cdot)$ operator is the elementwise multiplication of the vectors.

The structure of the LSTM language model is similar to the one in 2.2.1.1. But instead of directly using the embeddings after the projection layer, an LSTM layer is inserted between the output and the projection layer, as it is shown in Figure 2.4 [58]. The hidden states corresponding to the tokens in the input sequence are then used by the final output layer.

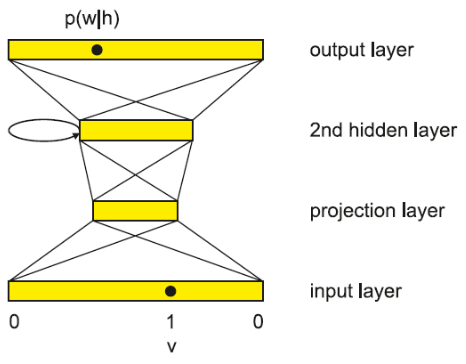


Figure 2.4: Language model with LSTM network used in 2nd hidden layer [58].

2.2.2 *Contextualized Embeddings*

Distributional word representations using static embeddings pre-trained on large scale unlabeled text corpora are a breakthrough for NLP systems [5, 37, 43]. It meant that the intrinsic features of textual data could be captured and represented in a certain feature space. In spite of that, these methods solely obtain a single global representation of each word, which effectively ignores its context. Contrary to the traditional static embeddings, contextual embeddings move beyond word-level semantics and associates each word with a representation that is a function of the entire input sequence [30]. In this subsection, two major contextualized embeddings that predate BERT are briefly introduced. This should serve as a precursor for a better understanding of the improvements that BERT brings.

2.2.2.1 *Embeddings from Language Models (ELMo)*

Peters et al. presented in their work [45] a deep language model, that could represent each token in a sequence as a function of the entire input sentence. This was made possible with the application of a bidirectional LSTM that was trained on a large text corpus with a coupled language model objective. Moreover, linear combinations of word vectors that were produced from the bidirectional LSTM could be stacked on top of each input word and be used for training for each end task. By combining the word vectors, rich word representations for the input tokens can be produced.

Similar to the objective function for a forward language model as presented in Equation 2.2, the authors in [45] created a backward language model, which predicts the previous token given the future context. Both the forward and backward language models are combined to produce a bidirectional model and the resulting log-likelihood objective function of Embeddings from Language Models (ELMo) to be maximized for the language modeling task can be presented as follows:

$$\begin{aligned}\sum_{k=1}^{N}(log\ p(t_k|t_1,...,t_{k-1};\Theta_x,\overrightarrow{\Theta}_{LSTM},\Theta_s)+\\ log\ p(t_k|t_{k+1},...,t_N;\Theta_x,\overleftarrow{\Theta}_{LSTM},\Theta_s))\end{aligned} \tag{2.13}$$

Parameters for the backward $\overleftarrow{\Theta}_{LSTM}$ and forward $\overrightarrow{\Theta}_{LSTM}$ LSTMs are seperated as shown in Equation 2.13, while the parameters for the token representation Θ_x, which is produced by the projection layer, and the parameter for the softmax layer $\Theta_s$are tied to both directions [45]. Hence, each token t_k receives $2L+1$ representations, $2L$ comes from the bidirectional LSTM and the last layer originates from the projection layer :

$$\begin{aligned}R_k &= \{x_k, \overrightarrow{h}_{k,j}, \overleftarrow{h}_{k,j}|j=1,....,L\}\\ &= \{h_{k,j}|j=1,....,L\}\end{aligned} \tag{2.14}$$

where R_k is the representation for each token, x_k is the token representation from the projection layer and finally $\overrightarrow{h}_{k,j}$, $\overleftarrow{h}_{k,j}$ are the representations from the forward and backward LSTM respectively. The weights for the bidirectional LSTM and the projection layers are pretrained on large text corpora before being used on downstream tasks such as text classification [31], question answering [49] and named entity recognition [44]. This step is called transfer learning, where a model is trained on domain specific data and relevant parts of the model are then taken to be applied on a downstream task that is relevant for the domain. By doing so, the bidirectional LSTM can compute representations for any task and these representations are context dependent due to the bidirectional feature that the LSTM layer provides. Subsequent fine-tuning of the model on task specific data then leads to an increase in performance for downstream tasks. For the application of ELMo on downstream tasks, ELMo flattens the layers in its representations R_k into a single vector. Generally, task specific weighting of all bidirectional LSTM layers are computed:

$$R_k^{task} = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j} \tag{2.15}$$

s_j^{task} are softmax-normalized weights for each layer in $h_{k,j}$ and the scalar parameter γ^{task} enables the task model to scale the entire ELMo vector [45]. The authors in [45] have also suggested that it may be helpful to apply a layer normalization to each bidirectional LSTM layer before the weighting, due to the activations in these bidirectional layers having different distributions.

2.2.2.2 *Contextualized Word Vectors (CoVe)*

Instead of using a large unlabeled text corpus to pretrain the word representations of a language model, McCann et al. used a deep LSTM attentional sequence-to-sequence model from a machine translation task to fine-tune on downstream tasks [36]. By pretraining the encoder on a machine translation task, the model is able to obtain contextual embeddings from the training data.

As shown by the authors in [36], pretraining the sequence-to-sequence model was done on English-to-German translations. According to their work, machine translation is an appropriate task for transfer learning because it generally requires the model to reproduce a sentence in the target language without losing information in the source language. The words were first tokenized and then embedded with Global Vectors (GloVe) embeddings [43]. The embeddings are then fed into an encoder, which consists of a bidirectional LSTM network, similar to the one in Subsection 2.2.2.1. Subsequently, the output from the bidirectional LSTM provides the context for the attentional decoder to produce a distribution over words in the target language, which in the case of [36] is German. The decoder itself is an unidirectional LSTM layer that receives three inputs: the target embedding from the previous time step, the context hidden state from encoder from the previous time step and also the hidden state of the previous time step from the same decoder [36]. The decoder computes a vector of attention weights, which expresses the importance of each encoding time-step to the current decoder state.

After pretraining the model, the encoder component of the model is transfered to downstream tasks to produce context vectors. Figure 2.5 shows the general pretraining and fine-tuning task done for CoVe. For classification and question answering tasks, the GloVe and CoVe embeddings of the input sequence are concatentaed before being used as features in the downstream task [36].

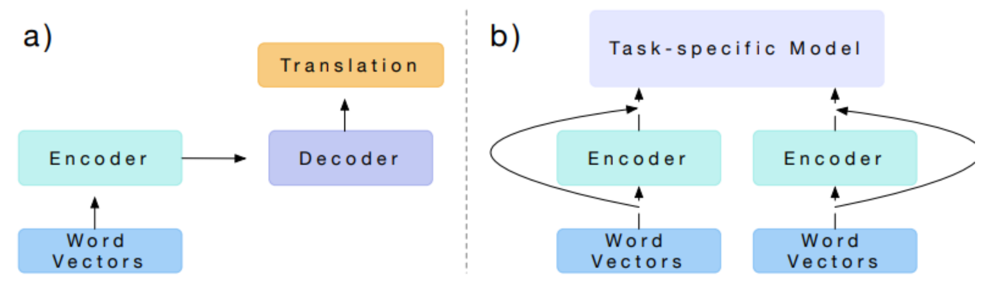


Figure 2.5: a) Pretraining of CoVe on a machine translation task. b) The encoder component from the pretraining phase is used as a feature extractor for downstream tasks [36].

2.3 TRANSFORMERS IN NLP

Previous methods used for extracting features and representations of words, such as the ones mentioned in Subsection 2.2.1 do not take into consideration the context of words or are unidirectional. On the contrary, the word embeddings used in the presented contextualized embeddings in Subsection 2.2.2 though bidirectional, require long pretraining periods, as data is fed sequentially into the model. This prohibits its ability to parallelize data input. These shortcomings were overcome when BERT was introduced by Devlin et al. in their work [13]. In this section, the attention mechanism, which is the fundamental principal that gave the transformer model its breakthrough is explained in Subsection 2.3.1. From there on, the transformer model is introduced and in the final subsection, the theoretical background behind the BERT model, which is the main model used in this thesis, is presented.

2.3.1 *Attention Mechanism*

For the implementation of a general model, it is imperative to first describe the general characteristics of a model that is able to employ attention. Such a model will be known as a task model in this subsection. At its most general form, the task model takes an input, carries a specified task and produces a desired output. Furthermore, the task model consists of four submodels: the feature model, query model, attention model and the output model [6]. The individual components of the task model can be seen in Figure 2.6 below:

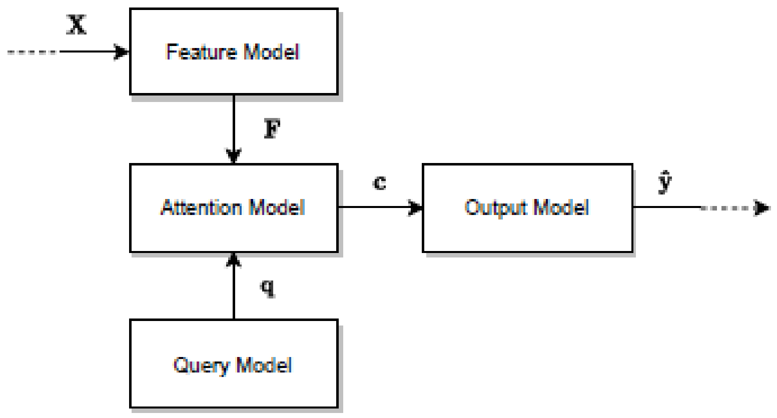


Figure 2.6: Task model for the application of attention mechanism. X is the input, F is a matrix of feature vectors , q is the query from the query model, c is the context vector and $\hat{y}$ is the output from the task model [6].

We follow [6] in illustrating the attention mechanism. Assume the task model takes in the matrix $X \in \mathbb{R}^{d_x \times n_x}$, where d_x represents the size of the input vectors and n_x represents the amount of input vectors. In the case of text input, n_x are the number of words or tokens in a text sequence [6]. A feature model is then applied to extract n_f feature vectors $f_1, ..., f_{n_f} \in \mathbb{R}^{d_f}$ from X. d_f could be the dimension size of a word embedding in the case of language models. To decide which feature vectors to attend to, the attention model needs a query $q \in \mathbb{R}^{d_q}$, where d_q represents the size of the query vector. The query is extracted through a query model and in general is constructed depending on the type of output that is desired by the model. The query can be interpreted literally as a question the model tries to ask [6]. For example, in the case of language modeling, one can produce an output of the target word to be predicted by using a query, that asks the question, which feature vectors from the input contains the necessary information to predict the subsequent words in a sentence.

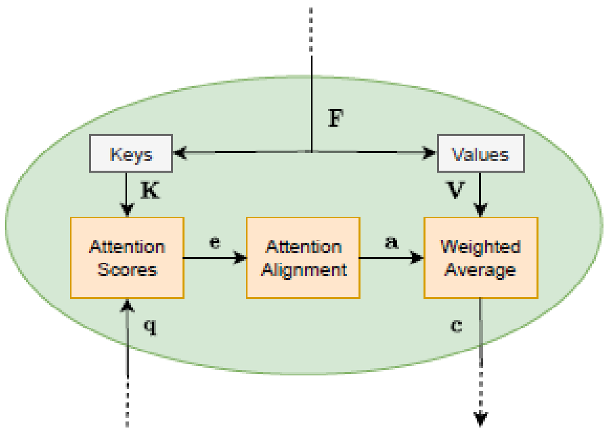


Figure 2.7: Inner workings of the attention model [6].

The attention model takes in the feature vectors as a matrix $F = [f_1, ..., f_{n_f}] \in \mathbb{R}^{d_f \times n_f}$ from the feature model and the query q from the query model. Two independent matrices are obtained from the matrix of feature vectors F. These are the keys and values matrix, denoted as $K = [k_1, ..., k_{n_f}] \in \mathbb{R}^{d_k \times n_f}$ and $V = [v_1, ..., v_{n_f}] \in \mathbb{R}^{d_v \times n_f}$, respectively. Generally, both these matrices are obtained by a linear transformation of F using weight matrices W_K and W_V for the keys and values matrix accordingly. These weights are learnable parameters during training or they can be predefined. These weights can either be used as identity matrices to retain the original feature vectors from the input or they can have different weights and dimensions to project the feature vectors into a new space. The only constraint that must be obeyed is that the number of columns in K and V must remain the same [6], as seen in the equations below:

$$\begin{aligned} K &= W_K \times F, \quad s.t. \quad K \in \mathbb{R}^{d_k \times n_f}, W_K \in \mathbb{R}^{d_k \times d_f}, F \in \mathbb{R}^{d_f \times n_f} \\ V &= W_V \times F, \quad s.t. \quad V \in \mathbb{R}^{d_v \times n_f}, W_V \in \mathbb{R}^{d_v \times d_f}, F \in \mathbb{R}^{d_f \times n_f} \end{aligned} \tag{2.16}$$

The objective of the attention model is to produce a weighted average of the value matrix V [6]. These weights used to produce the output are computed by calculating the attention score and executing an alignment step. The query and the keys matrix are used to calculate the attention score vector $e = [e_1, ..., e_{n_f}] \in \mathbb{R}^{n_f}$. More specifically, the attention scores in the attention matrix represents how important information contained in the keys vector of the keys matrix is according to the query [6]. The calculation for the attention score is done via a score function $score(.)$:

$$e_l = score(q, k) \tag{2.17}$$

Here, $e_l \in \mathbb{R}$ represents lth value in vector e. In [62], the attention scoring function used for the transformer is a scaled dot product, which is essentially a cosine-similarity function [14]. The attention scores are redistributed via an alignment function, as the objective is to produce a weighted average of the values vector. It also allows the attention scores to be constrained within a range inside of $[0, 1]$. One example of an alignment function is the softmax function, as shown in Equation 2.6. As each weight is a direct indication of how relevant each feature vector is to the others, it provides a deeper understanding of the model's behaviour and its relationships between the outputs and inputs. The application of the alignment function is shown in the equation below:

$$a_l = align(e_l, e) \tag{2.18}$$

$a_l \in \mathbb{R}$ corresponds to the attention weight for the lth value. The vector of attention weights $a = [a_1, ..., a_{n_f}] \in \mathbb{R}^{n_f}$ is multiplied with the values vector to produce the context vector $c \in \mathbb{R}^{d_v}$, which is the contextualized embedding in the case of language modeling. The output model in 2.6 utilizes the context vector to output a prediction. The equations below summarize the procedure of computing the context vector c and the output prediction $\hat{y}$ as depicted in Figure 2.6.

$$c = \sum_{l=1}^{n_f} a_l \times v_l \tag{2.19}$$

$$\hat{y} = softmax(W_c \times c + b_c) \tag{2.20}$$

In Equation 2.20, $W_c \in \mathbb{R}^{d_y \times d_v}$ and $b_c \in \mathbb{R}^{d_y}$ are the weights and biases for the output model. d_y represents the number of output classes. In the language modelling setting, d_y would be the same as the size of the vocabulary and the output is used to predict the probability of the next word based on the context of the sentence.

2.3.2 *Transformer architecture*

The attention mechanism gained traction in the NLP field, especially with the introduction of transformers [14, 62] for sequence-to-sequence annotation. According to [62], the transformer is the first sequence-to-sequence model that relies on self-attention to compute representations of the input and output, without using sequence-aligned RNNs. This intrinsically motivates parallelization within training examples, which becomes critical when processing longer sequence lengths.

The transformer's architecture follows most state of the art neural sequence transduction models, which have an encoder-decoder structure. The encoder projects an input sequence of representations to a sequence of learned contextualized embeddings. Given these representations from the encoder component, the decoder component of the model generates an output sequence one element at a time. By consuming the previously generated element as additional input when generating the next, the model is said to be autoregressive [62]. The architecture of the transformer can be summarized in the following figure:

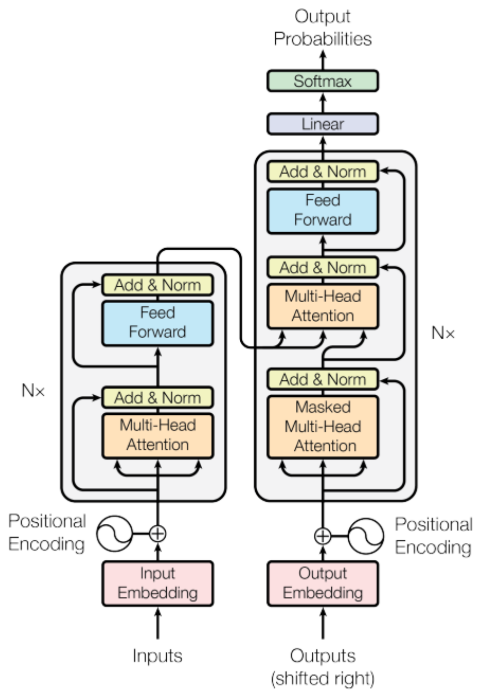


Figure 2.8: Architecture of a transformer model [62].

The encoder consists of a stack of $N = 6$ identical layers and each layer has two sublayers, through which the input is subjected to. The first sublayer is a multi-headed self-attention mechanism and the second is a position-wise fully connected FFNN. Residual connections, which are followed by layer normalizations, are employed on each operation to prevent the vanishing or exploding gradient problem in the training of a deep neural network [60]. Similar to the encoder, the decoder component consists of a stack of $N = 6$ identical layers. Along with the two sublayers in the encoder layer, the decoder layer takes in a representation of the output and also the output from the encoder layer stack to perform multi-head attention. As it can be seen in Figure 2.8, residual connections followed by layer normalization are employed over each sublayer. The first multi-head attention sublayer of the decoder in Figure 2.8 are modified by masking to prevent positions from attending to ensuing positions, hence preserving the autoregressive property [62]. The masking of output embeddings together with the fact that the output embeddings are offset by one position, enables the predictions for a certain position to only depend on the known outputs at the preceding positions.

Due to the abscence of recurrence and convolution, the model cannot make use of the order of the sequence without a positional encoding. The additional information about the relative positions of the tokens in sequence is hence inserted into the input embeddings by Vaswani et al. in the transformer architecture. The positional encodings have the same dimensions as the input embeddings, which makes the summation of both these embeddings possible. In [62], sine and cosine functions of different frequencies were used to capture the positional information of the tokens (Refer to Equation 2.21). pos is the absolute position of token in sequence and i is the dimension of the embedding. To this end, each dimension of the positional encoding is represented by a sinusoid. This method of encoding positions of tokens was chosen by the authors in [62], as it allowed the model to extrapolate to sequence lengths longer than the ones used in training. A visual example output of the positional encodings with a dimension size of 128 is shown in Figure 2.9

$$\begin{aligned} PE_{pos,2i} &= sin(pos/10000^{2i/d_{model}}) \\ PE_{pos,2i+1} &= cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{2.21}$$

As discussed in Subsection 2.3.1, the transformer architecture also utilizes the attention mechanism and applies a scaled dot product as the score function. The dot product essentially computes the cosine similarity between the query and key by evaluating the dot product between the query and key matrices. The dot product between query and key matrices is then subjected to a softmax function and is subsequently scaled by the hidden dimension of the key matrix. Applying the softmax function aligns the attention scores, as mentioned in Subsection 2.3.1, and scaling is done to prevent the softmax function from having extremely small gradients [62]. Both key and query matrices have the same hidden dimension size in [62]. Since the dot product can be implemented using optimized matrix multiplication code, the scaled dot product is in practice fast and space-efficient. A specialized variation of the attention mechanism is used for the transformer model which is known as self-attention. Similar to the attention mechanism in Subsection 2.3.1, a query model is used to extract a query matrix to calculate the attention score. But instead of using another input for the query model, the query is derived from the feature vectors. This is done similar to the calculation of the keys and values matrix in Subsection 2.3.1, where the query matrix is obtained by a linear transformation using a learnable weight matrix in a deep neural network setting. Hence, the query of the model is learned along the way during training and is used to uncover relations between feature vectors [6]. The relations uncovered by self-attention can be used as supplementary information to incorporate into new representations for the feature vectors and therefore summarizes the information that is relevant to the query. The self-attention implemented in the transformer architecture is shown in Equation 2.22. Equation 2.22 thus summarizes Equations 2.17, 2.18 and 2.19.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}}) * V \tag{2.22}$$

Rather than performing a single attention function with the entire dimension size of queries, keys and values, Vaswani et al. discovered that it was more practical to linearly project the queries, keys and values h times with different learned linear weights to d_q, d_k and d_v, respectively. The dimension size d_k is the same as the size d_q in order for the dot product to be calculated. This effectually splits the original dimension size of the linear weights h times and therefore allows for the parallelization of the attention function. The splitting of these linear weights also promotes the model to learn different attentions on the feature vector, through which different relations between feature vectors can be calculated [62]. For example, relations between words such as: which verbs refer to which nouns, which nouns are being referred by the pronouns, etc. Once all h context vectors have been calculated in parallel, they are concatenated and once again projected with a weight matrix (Refer to Equation 2.23).

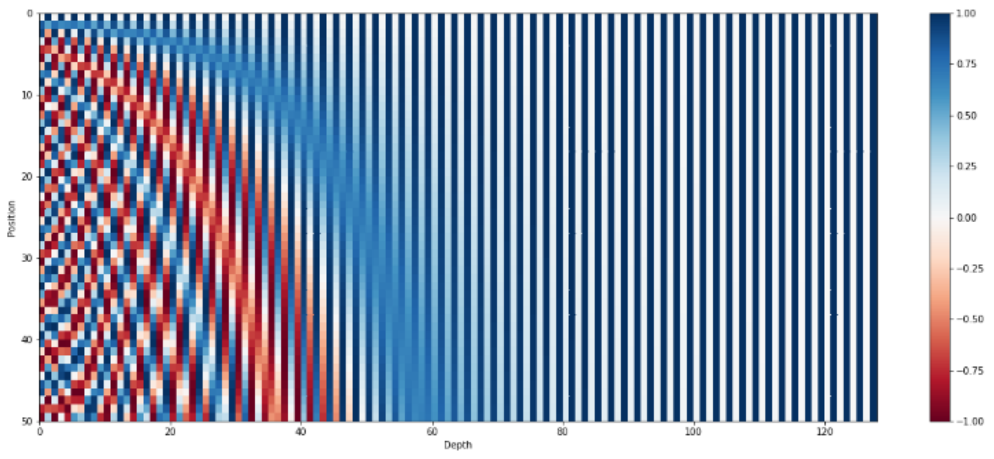


Figure 2.9: Positional encodings represented in each dimension. Position on the y-axis represents the absolute position of token in sequence and depth on the x-axis corresponds to the dimension. A dimension size of 128 was used in this figure [25].

$$\begin{aligned} &MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^0 \\ &where \quad head_i = Attention(Q_i, K_i, V_i) \end{aligned} \tag{2.23}$$

$W^0 \in \mathbb{R}^{hd_v \times d_{model}}$ in the above equation is the output weight matrix that produces an output of dimension d_{model}. It should also be mentioned here that the second sublayer of the decoder in the transformer model as shown in Figure 2.8 accepts queries from the previous decoder layer and the keys and values originate from the output of the encoder. This enables every position in the decoder to attend to all positions in the input sequence from the encoder [62].

2.3.3 *Bidirectional Encoder Representations from Transformers (BERT)*

With the introduction of the transformer architecture, causal language models became a standard in sequence-to-sequence tasks. Though using causal language models is recommended for tasks such as text generation and machine translation [34], other tasks only require hidden features of texts to be extracted. To this end, the model architecture of BERT, which is based on the original transformer model as discussed in Subsection 2.3.2, consists of a multi-layer bidirectional Transformer encoder. BERT does not contain the decoder component as in the transformer architecture. Two variants of BERT were introduced in [13]: **BERT$_{\text{BASE}}$** and **BERT$_{\text{LARGE}}$**. **BERT$_{\text{BASE}}$** has 12 encoder layers, denoted by L, 12 attention heads h, as described in Subsection 2.3.2 and a dimension size d_{model} of 768 for the feature vector, which is also known as the hidden dimension size of the model. Whereas **BERT$_{\text{LARGE}}$** has 24 encoder layers, 16 attention heads and a d_{model} of 1024.

The two important steps in training and implementing BERT are the pre-training and fine-tuning. In the course of pre-training, the model is trained in an unsupervised fashion on unlabeled text data over different pre-training tasks. Whereas for fine-tuning, the BERT model is firstly loaded with the pre-trained weights and all the parameters are fine-tuned using labeled data in downstream tasks [13]. The pre-training corpus used for pre-training **BERT$_{\text{BASE}}$** and **BERT$_{\text{LARGE}}$** was the BooksCorpus from [68] with 800M words and the English Wikipedia corpus with 2500M words. Two pre-training methods were employed by Devlin et al. for BERT: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). During MLM, 15% of the input tokens are masked at random and the objective is to predict the masked tokens, similar to the objective function in 2.7. Of those 15% masked tokens, 80% of the time the word is replaced with a special [MASK] token, 10% of the time the word is replaced with a random token and for the remaining 10% , the token is unchanged. By having the token unchanged for 10% of the time, the input representation is biased towards the actual observed word [13]. The advantage of this pre-training process is that the encoder does not know which words it will be asked to predict and therefore forces the encoder to keep a distributional contextual representation of every input token [13]. In contrast to MLM, NSP helps for downstream tasks where sentence level prediction such as question answering and natural language inference is the objective. In the case of NSP, the model receives two sentences and the goal is to predict if the second sentence is an actual sentence that follows the first input sentence. For each downstream task there are different models, which are initialized with the same pre-trained parameters and for practical purposes, the downstream tasks should be in the same domain as the unlabeled data used for pre-training the model.

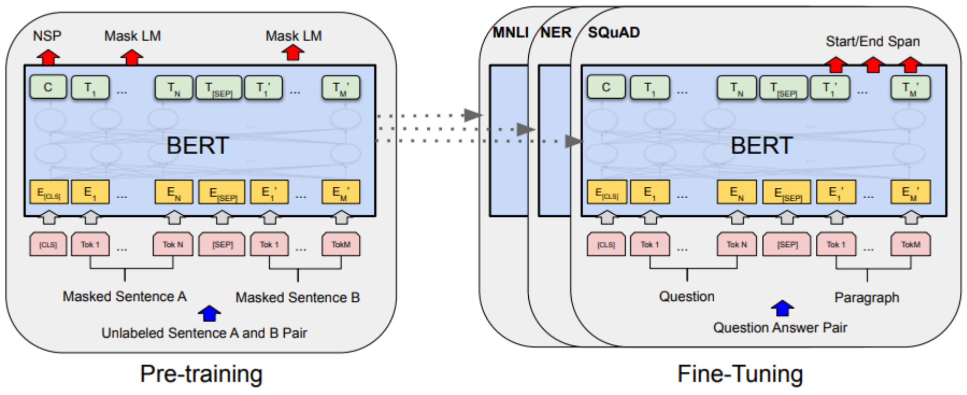


Figure 2.10: Pre-training and fine-tuning operations for BERT [13].

In order for BERT to operate on a variety of downstream tasks, the input embeddings should be able to easily represent both a single sentence and a pair of sentences. An example case where a pair of sentences are needed for fine-tuning is the downstream task of question answering, where one sentence corresponds to the question and the other sentence corresponds to the answer. The term "sentence" refers to an arbitrary span of adjoining text, instead of an actual linguistic sentence. The input embeddings of the BERT model is made up of three embeddings, namely the token embeddings, the segment embeddings and position embeddings as shown in Figure 2.11.

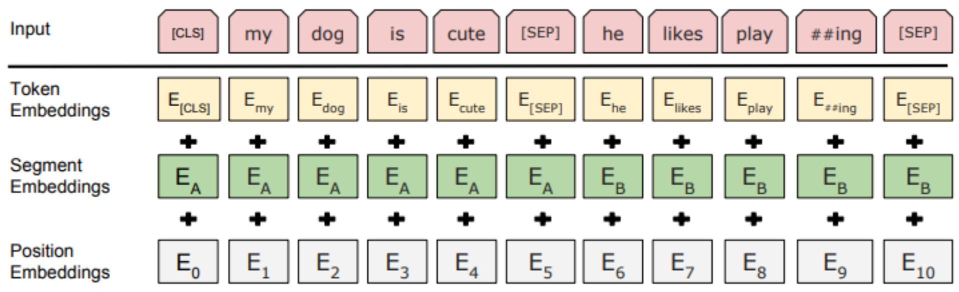


Figure 2.11: Input representations of BERT made up of token embeddings, segment embeddings and position embeddings [13].

The token embeddings require that the input sequence be first tokenized with a method known as WordPiece tokenization [55]. Given a text input, WordPiece first tokenizes the text into words by splitting on punctuation and whitespaces. Then it tokenizes each word into subword entities known as wordpieces. By doing so, the WordPiece tokenizer can handle out-of-vocab words that are not found in the vocabulary used for BERT. For example, the word "algebra" may not be in the vocabulary used for tokenizing text sequences in BERT, hence the tokenizer looks for the next longest subwords that makes up the word "algebra" and tokenizes the word into the wordpieces "al", "##ge", "##bra". The prefix "##" is used in the WordPiece tokenization process to signal that the wordpiece belongs to the preceeding wordpiece in the sequence. As shown in the summary of embeddings that make up the input representation in Figure 2.11, the word "playing" is seperated into two tokens: "play" and "##ing". It is also interesting to note, that the token embeddings add special tokens at the beginning and at the end of sentences. The [CLS] token stands for a classification token and is used to represent the sentence-level representation of the sentence, whereas the [SEP] token is used to seperate the first input sentence from the second [13]. The [SEP] token also signifies the end of a sentence. A custom WordPiece tokenizer can be trained to build up a vocabulary for tokenization. This would prove useful when the original BERT vocabulary used in **BERT$_{\text{BASE}}$** and **BERT$_{\text{LARGE}}$** does not cover most of the vocabulary used in the task at hand or when tokenizing characters from a foreign language. The tokenizer is trained in a similar manner to the Byte-Pair Encoding tokenizer [53]. The vocabulary of the tokenizer is first initialized with all the characters present at the beginning of a word and the characters present within a word in a given corpus. Similar to Byte-Pair Encoding, WordPiece learns merge rules. WordPiece finds the most frequent pair of adjacent tokens and calculates a score, given by Equation 2.24 [21]. The adjacent pair of tokens with the highest aforementioned score are merged and inserted as a new token in the vocabulary. This process is repeated until a predetermined size of vocabulary is reached. Dividing the frequency of the newly formed pair by the product of the frequencies of its substituents helps the algorithm in prioritizing the merging of pairs where the individual parts are less frequent in the vocabulary.

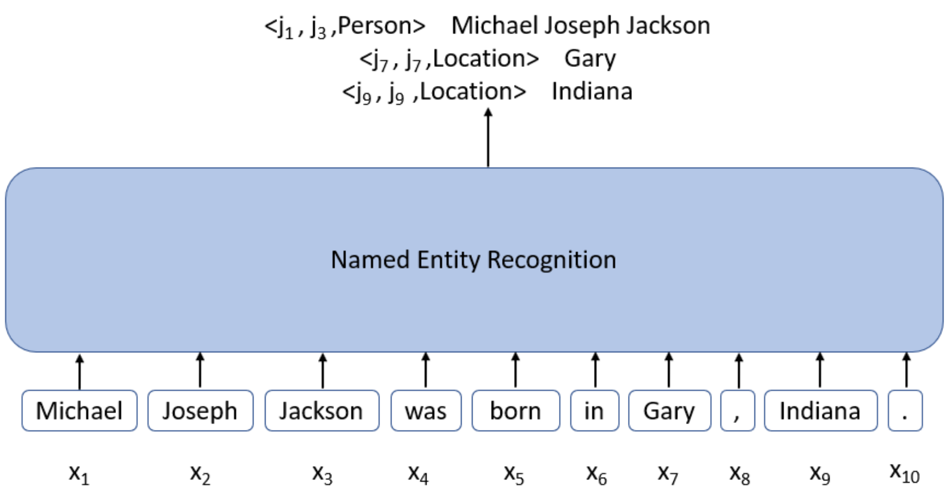
$$\frac{Frequency\_of\_pair}{Frequency\_of\_first\_element \times Freqeuncy\_of\_second\_element} \tag{2.24}$$

The vocabulary size of **BERT$_{\text{BASE}}$** and **BERT$_{\text{LARGE}}$** is 30522. Devlin et al. had also built 2 variants of the vocabulary, one cased and the other uncased. The cased vocabulary tokenizes words as is by not converting the tokens to lowercase, whereas the uncased vocabulary converts tokens that are capitalised to lowercases. By tokenizing the words in a text sequence and assigning the words token ids, each token can then be represented by a token embedding of size d_{model}. For **BERT$_{\text{BASE}}$** this dimension size is 768 and for **BERT$_{\text{LARGE}}$** it is 1024. The segment embeddings assign tokens that belong to one sentence an index 0 and assigns tokens of the other sentence a different index of 1. Through this embedding, the model can differentiate which tokens belong to which sentence. Lastly, the position embeddings used for BERT is similar to the one used in the transformer architecture, as described in Subsection 2.3.2. Both segment embeddings and position embeddings have an embedding size of d_{model}, similar to the token embedding.

2.4 NAMED-ENTITY RECOGNITION (NER)

In the field of data mining, a named-entity can be defined as a word or a phrase that clearly distinguishes one item from a set of other items that have analogous attributes [29]. The word "named" narrows down the scope of entities in a sentence that have one or many rigid designators that semantically refer to the same object. A rigid designator is an expression that refers to the same object or entity in every possible world. Although rigid designators usually comprise of proper names, they also depend on the domain of interest. some examples of named-entities are person, location and organization in the general domain; whereas organic compounds, organic polymers and chemical reactions are more suited as entities of interest in the chemical field.

Given a sequence of tokens $s = x_1, ..., x_L$, a NER-model outputs a list of tuples $\langle j_s, j_e, E \rangle$, with each tuple representing a named-entitiy in s. Here, $j_s \in [1, L]$ and $j_e \in [1, L]$ are the start and end indexes of a named-entity instance, correspondingly. E is the entity type which is predetermined by either the model or a dataset. Figure 2.12 illustrates an example output of a named entity recognition system. NER-tasks can be further broken down into coarse-grained NER and fine-grained NER, where the former assigns a single entity type for each named-entity detected and the latter allows a named-entity instance to be allocated multiple entity types. In this work, the course-grained NER datasets are to be the focus.



<j1, j3 ,Person> Michael Joseph Jackson
<j7, j7 ,Location> Gary
<j9, j9 ,Location> Indiana
Named Entity Recognition
Michael
Joseph
Jackson
was
born
in
Gary
,
Indiana
.
x1 x2 x3 x4 x5 x6 x7 x8 x9 x10

Figure 2.12: An example of a named entity recognition system.

2.5 CONDITIONAL RANDOM FIELDS (CRF)

Though often times features extracted from language models such as BERT can be used directly in a linear neural network layer to return output probabilities per label in a NER task, this naïve procedure often assumes that each prediction for a specific token is independent of the predictions of the neighbouring tokens. This could pose a challenge for a task that implicitly requires predictions to be aware of the predictions of surrounding neigbour tokens and hence imply a certain pattern of order for predictions. For an instance, the B-tag in an IOB2-scheme explained in 3.1.3 should always come before an I-tag of the same entity type when an entity is detected and spans over more than one word. Originally proposed by [26] for labeling sequential data, the CRF has expanded its application in NLP where sequence labeling tasks are needed to be solved, such as the case for NER and Parts-of-Speech (POS) tagging.

The most common form of CRF in sequence labeling tasks is the linear chain CRF, as these are more tractable during inference. Given an input sequence of representations of tokens $X = (x_1, ..., x_L)$ and a sequence of token predictions $Y = (y_1, ..., y_L)$ and $y_i \in \{1, ..., K\}$, where K is the number of prediction tags, the linear chain CRF computes a score of the sequence which is defined as:

$$s(X, Y) = \sum_{i=1}^{L} E(x_i, y_i) + \sum_{i=1}^{L-1} T(y_i, y_{i+1}) \tag{2.25}$$

$E(x_i, y_i)$ denotes the emission score and signifies the score allocated to each prediction y based on the input representation x after i iterations. The input representation can contain any type of information, but typically contains information on surrounding tokens and its context [1]. The term $T(y_i, y_{i+1})$ denotes the transition score and assigns the score when the model prediction transitions from position i to $i+1$ in the same sequence. Specifically, it signifies the chance that the prediction y_{i+1} comes after y_i. Hence, they only impose dependencies on the previous prediction. To make use of this score and compute the conditional probability $p(Y|X)$, a partition function $Z(X)$ is introduced:

$$\begin{gathered} P(Y|X) = \frac{e^{S(X,Y)}}{Z(X)} \\ \textit{where} \\ Z(X) = \sum_{\tilde{y} \in Y_X} e^{s(X,\tilde{y})} \end{gathered} \tag{2.26}$$

Y_X are all possible prediction tag sequences. $Z(X)$ can be considered a normalization function to discover a distribution of probabilities [1]. Due to the recurrent properties of $Z(X)$, computing its values is not a simple task, as all iterations of the input for each step has to be considered. Despite the fact that the complexity of calculating $Z(X)$ is very high ($O(|y|^k)$), its recurrent property also allows dynamic programming to solve its summation, which then reduces its complexity down to $O(k * |y|^2)$. This can be achieved with the forward/backward algorithm [1, 26]. During training, the goal is to minimize the negative log-likelihood of the prediction sequence. Where the forward/backward algorithm is used to ease the computation of $Z(X)$, the Viterbi algorithm is used to determine the most likely sequence with the highest $P(Y|X)$ during inference [57, 59].

It would be of special interest to investigate, whether the performances of the domain-specific models used in this work on the NER tasks can be improved with the help of linear chain CRF. To study its effects, the BERT models pre-trained in this work are fitted with a linear neural network layer that projects the hidden dimension of the token representations to the label space. The output of the linear neural network layer is subsequently passed to a CRF layer.

2.6 RELATED WORK

Because unsupervised training of language models on large corpora has been shown to improve on many NLP tasks [13, 45], there has been a recent trend on pre-training language models on domain specific texts to improve performance on a variety of NLP tasks that require domain-specific input. To this end, a few examples of pre-trained domain-specific or otherwise different from general text corpus BERT models that are related to a scientific field are introduced in this subsection. There exists in addition to that other domain-specific BERT models such as FinBERT [2], LEGAL-BERT [7] and PatentBERT [27], but these are not discussed here as they are not related to scientific fields.

The first reference of language models pre-trained on scientific text corpora was introduced by Beltagy, Lo, and Cohan [4]. The authors pre-trained the model on a random sample of 1.14M papers from Semantic Scholar, where 82% originate from the broad biomedical domain and and the remaining 18% come from the computer science domain [4]. A total of 3.17B tokens were used for training, which is similar to 3.3B tokens on which BERT was pre-trained on in [13]. Besides that, the authors also introduced a custom vocabulary for its tokenizer called SCIVOCAB, which is a new WordPiece vocabulary used on the scientific pre-training corpus and it was trained with the SentencePiece library [54]. SCIVOCAB managed to capture 58% of its vocabulary from the pre-training corpus which do not exist in the vocabulary used by BERT's tokenizer [4]. **SciBERT** underwent the same pre-training procedure as BERT with MLM and NSP. The performance of **SciBERT** was evaluated on five main NLP tasks: NER, PICO extraction, text classification, relation classification and dependency parsing. PICO extraction is similar to NER, where the model detects and classifies token spans into four classes, namely Participants, Interventions, Comparisons and Outcomes in a clinical trial paper [4]. **SciBERT** managed to perform better than the state of the art results for NER, text classification and relation classification for the computer science domain at the time of publication, achieving an increase in macro F1 score of 3.55 with fine-tuning and an increase of 1.13 without fine-tuning [4]. Simultaneously, the model had also achieved state of the art results in NLP tasks of the biomedical domain.

During the same period, when Beltagy, Lo, and Cohan presented **SciBERT**, a pre-trained biomedical language representation model BioBERT was introduced in [28]. The model was pre-trained on PubMed abstracts and PubMed Central full-text articles, each with 4.5B and 13.5 B words respectively [28]. Instead of constructing a new vocabulary for the tokenizer, BioBERT used the readily available cased vocabulary of BERT. This is to allow BioBERT to be compatible with BERT. During fine-tuning, BioBERT was trained for three biomedical text mining tasks which were NER, relation extraction and question answering. BioBERT managed to outperform BERT and other earlier state of the art models in biomedical text mining tasks. BioBERT outperforms BERT on biomedical NER by a F1 score of 0.62% and biomedical relation extraction by 2.80% [28]. Although BioBERT was pre-trained on purely biomedical text corpora, **SciBERT** was able to perform better than BioBERT in 2 NER datasets and one dataset for relation extraction task [4]. This could be due to **SciBERT** employing its custom vocabulary which can represent more words in the biomedical domain.

Huang, Altosaar, and Ranganath presented ClinicalBERT to build representations of clinical notes during a patient's admission in order to predict the patient's risk of being readmitted within a 30-day time frame [18]. The objective functions for pre-training ClinicalBERT were similar to the ones used in [13]. During fine-tuning, the embeddings from ClinicalBERT were used to train for a downstream binary classification task for readmission prediction. In contrast to previous works on domain-specific BERT models, the authors of [18] also tested ClinicalBERT by evaluating its ability to rank similarity scores of 30 pairs of medical terms. This is done by computing embeddings of a medical term and the sum of the last four hidden states of ClinicalBERT of the encoder layers was used to represent the medical term. This practice however yields results that are not credible as this method is often worse than averaging GloVe embeddings of the input [48]. ClinicalBERT outperformed BERT on the downstream task.

Besides the medical field, BERT started gaining popularity within the material science domain. The first usage of a domain-specific BERT for material science came with the introduction of MatSciBERT [16]. Research papers from the domain were searched on Crossref and the papers were downloaded from the Elsevier Science Direct database. The text from these research papers were then extracted with the help of a custom XML parser and the extracted texts were preprocessed, where random Unicode characters were removed and different Unicode characters having similar meaning and appearance were replaced with single standard characters. Different from previous mentioned works on domain-specific models, the authors of [16] initiated the weights for BERT with the weights from **SciBERT** to pre-train on their text corpus. The authors also used the SCIVOCAB to tokenize the input sequences for the reason that the uncased SCIVOCAB could cover 53.64% of the material science vocabulary, whereas the vocabulary from BERT was only able to cover 38.90% [16]. Besides that, it is more practical than building a custom vocabulary for the material science domain, which would cause pre-training to take a longer time. MatSciBERT also went through a different approach for pre-training its model. Instead of using both MLM and NSP, the authors in [16] employed a whole word masking method, where the masking of tokens is done on a word level and tokens that belong to a word are masked all together. MatSciBERT performed better than **SciBERT** in all downstream tasks in the material science domain, namely NER, text classification and relation classification. Another similar BERT model pre-trained on a material science corpus was also included in the research of MatBERT in [61]. Pre-training for MatBERT was done on two million papers which were randomly sampled from a corpus of peer-reviewed material science journal articles. Trewartha et al. employed two custom vocabularies to train its tokenizers. One of these vocabularies is cased and the other uncased. Furthermore, the objective of pre-training was solely MLM. In contrast to MatSciBERT, MatBERT was pre-trained by initiating its BERT model with BERT weights instead of **SciBERT** weights. MatBERT achieved an overall 1% to 4% improvement over **SciBERT** on downstream NER tasks. It was also discovered in [61], that **SciBERT** outperformed BERT on the same downstream NER tasks by 3% to 9%, which emphasizes the importance of scientific pre-training.

Another example of a custom pre-trained language model used for domain-specific purposes was shown in [19]. In the work of Huang and Cole, the authors constructed 6 variants of domain-specific BERT models for text mining in the field of battery research and the models were used to update and enhance a battery database that was generated using ChemDataExtractor [20]. Two of which were initiated with BERT model weights (BatteryBERT), another two variants were initiated with **SciBERT** model weights (BatterySciBERT) and the remaining two were trained from scratch with a BERT architecture (BatteryOnlyBERT). In each pair of variants initiated with the same model weights, one model used a cased vocabulary and the other used an uncased vocabulary. When comparing BatteryOnlyBERT with BatteryBERT as well as BatterySciBERT in the downstream NER task, BatterySciBERT and BatteryBERT perform better than BatteryOnlyBERT when it came to precision and recall as evaluation measures. This could be due to the fact that the number of training steps for BatteryOnlyBERT was not sufficient. Generally, BatteryBERT, BatterySciBERT as well as BatteryOnlyBERT outperformed the original **BERT$_{\textbf{BASE}}$** in the downstream task of NER. Moreover, BatteryBERT was employed to enhance a battery database by [20] by building in a document classifier that filters out documents not related to battery research and also a question answering agent in its data extraction pipeline. The pre-programed questions about various device materials in the question answering agent allowed the pipeline to classify device components and be saved in the battery database.

3 EMPIRICAL STUDY

In this section, the datasets as well as the methodologies used in this work are described. Firstly, the origins of the text corpus from the physics and computer science domain for pre-training is reported. Furthermore, the NER datasets used for evaluating the BERT language models pre-trained in this work is introduced. For both the text corpus as well as the NER datasets, data preprocessing procedures were executed, which are also described in their respective sections. The methodologies used for pre-training as well as fine-tuning on the downstream task of NER are also explained.

3.1 DATA

3.1.1 *arXiv Text Corpus*

In order to investigate how well the present BERT model architecture can represent texts from the physics and computer science domain, a text corpus from these two domains was built in this thesis. Text data from these two domains were extracted by first downloading research articles from the open access repository arXiv. The physics domain is further broken down into different subcategories, which are described in Table A.1. The period of publication of the aforementioned research articles are from 01. September 1998 until 10. December 2021. For bulk downloads, a Python script provided by arXiv was used to download the papers from a cloud storage service [69]. A total of 1,580,695 papers were downloaded as PDF files. From this number of papers, text data from 1,560,661 research papers were successfully extracted. Approximately 29% of the extracted texts originated from the computer science domain and the rest comes from the physics domain. The extraction and parsing of the textual data was carried out with the help of a Python binding for lightweight PDF, XPS and E-book viewing and rendering called PyMuPDF [46]. During extraction and parsing, the texts were stored in text files and encoded with the UTF-8 encoding. Unencodable Unicodes are replaced with a question mark. The remaining 20034 papers could not be used with this Python library due to errors from non-standardized font sizes and corrupted downloads from arXiv.

After the extraction of texts from downloaded research papers, the text data was cleaned with a series of data preprocessing procedures. Firstly, similar to previous works mentioned in 2.6, the figures and tables from the research articles were removed, as they do not contain meaningful textual information. Captions of figures and tables were retained, as these contain important textual information. At the same time, sentences that contain a high ratio non-alphabetic characters such as mathematical formulas and table rows were removed. A procedure similar to the preprocessing procedure in [4] was used. This procedure removes sentences that did not fulfill all of the following requirements:

- The sentence must have at least 4 tokens after being tokenized by the tokenizer.
- The ratio of tokens that solely contain alphabets to the total number of tokens must be more than or equal to 0.4.
- The ratio of characters that are alphabets to the total number of characters in the sentence must be more than or equal to 0.6.

During the extraction of sentences, incomplete sentences were also extracted especially when the texts in the research papers were formatted in a multi-column structure. To alleviate this issue, sentences which have an alphabetic character in upper case at the beginning and have a total character length of more than 80 characters as well as end with a full-stop were not removed. The predetermined total character length of 80 is arbitrary. However, this allowed short sentences like titles of subsections as well as sentences that only contain abbreviations that end with full stops such as "Fig." to be removed. Other forms of end of sentence punctuation such as question marks and exclamation marks were not considered, as they could also be used in mathematical formulas that are written within the sentence. Moreover, it was rare to encounter both of these punctuations as end of sentence punctuation in the context of scientific research papers. Texts in scientific literature also contain symbols and random characters. To address this, the texts also underwent a normalization procedure, similar to the one employed in [16]. The process involved normalizing the corpus with BertNormalizer from the Hugging Face tokenizers library [21]. The BertNormalizer strips accents from characters and also remove any control characters and replaces all whitespaces with the classic whitespace. Thereafter, a list of mappings of Unicode characters created by [16] was used to map different Unicode characters having similar definitions and appearance to either a single standard Unicode character or a sequence of standard characters. To create a training corpus that is representative of the overall text corpus, the lines in the training corpus was shuffled randomly. After all the pre-processing procedures, the final pre-training corpus produced had a size of 16GB and an evaluation corpus, which is needed for the measurement of its pseudo-perplexity in Subsection 3.2.1, with a size of 81MB was also created.

In order to gain insight into the text corpus used for pre-training, the training corpus was explored by counting unique its unique words and also training a WordPiece tokenizer like the one used in [13] and as described in Subsection 2.3.3. Unique words which were counted did not include common stop words such as "the", "is" and "and" which could mislead the word count as these words are very common among texts. The stop words for filtering out the words were taken from the Python library NLTK [40].

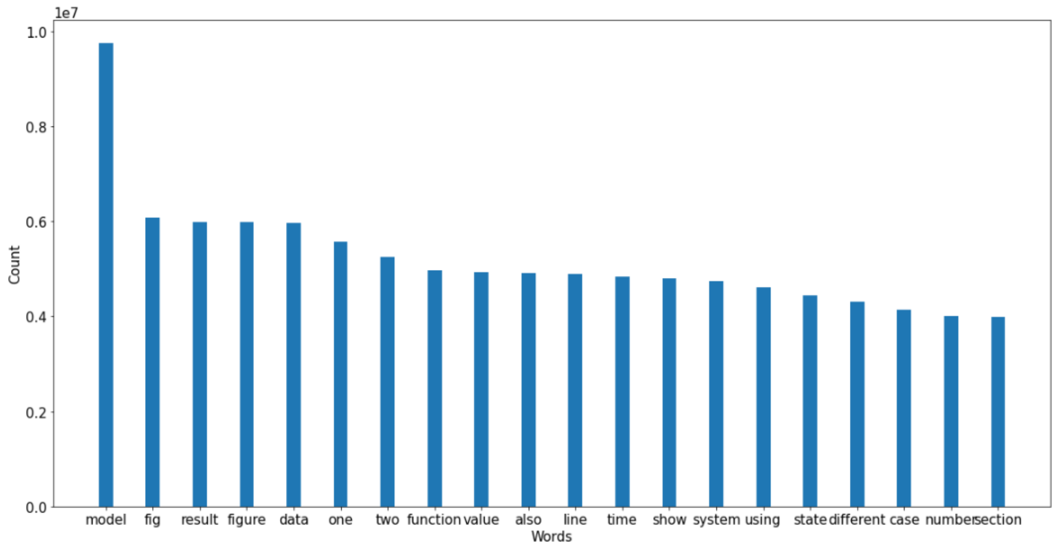


Figure 3.1: Unique words in pre-training corpus for training without including stop words.

As it is shown in Figure 3.1, the word "model" appears most frequently in the text corpus, followed by "fig", which is commonly used as an abbreviation in research articles for the word "figure", and "result". A comparison of frequently occurring words in our corpus with the corpora for pre-training **BERT$_{\textbf{BASE}}$** [13] as well as **SciBERT** [4] is also shown in Figure 3.2. The words present in our training corpus are similar to that of **SciBERT** than that of **BERT$_{\textbf{BASE}}$** as less scientific terms were captured in the general domain of BERT. It can then be assumed that initializing the weights for pre-training with the weights of **SciBERT** should provide better results.

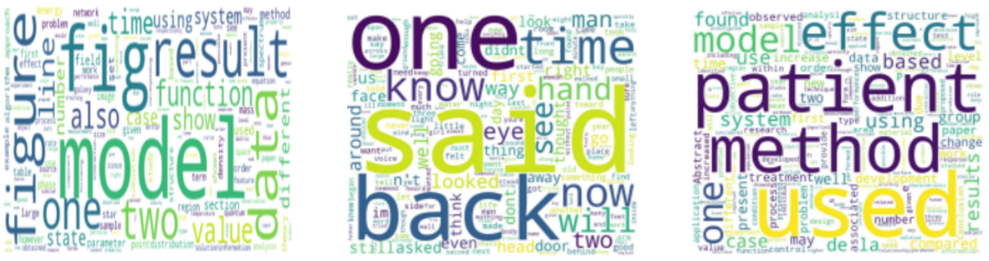


Figure 3.2: Wordcloud comparison for our corpus (left), **BERT$_{\text{BASE}}$** (middle), **SciBERT** (right). Wordclouds for **BERT$_{\text{BASE}}$** and **SciBERT** taken from [19].

Training the WordPiece tokenizer in our corpus also gives further insight into the number of new distinct tokens or subwords existing in the vocabulary of the texts extracted (refer to Subsection 2.3.3). Figure 3.3 shows the Venn diagrams for the vocabulary produced after training the WordPiece tokenizer on our text corpus extracted from arXiv and also the vocabularies from the **BERT$_{\text{BASE}}$** and **SciBERT** models. Both cased and uncased variants of the vocabularies are shown in Figure 3.3. Planned comparisons between all three training data of the models for pre-training revealed that the uncased vocabulary share more similar tokens than the cased vocabulary. As there are more unique words when words are distinguished by their casings it is expected that the uncased vocabularies of these models would share a higher number of words. Evaluating the similarities of the vocabulary between the models also yields that the training data used for **SciBERT** share a higher number of words or subwords in its vocabulary (6174 for uncased, 7123 for cased) with the training data used in this work than **BERT$_{\text{BASE}}$** does (2251 for uncased, 1626 for cased). This shows that the **SciBERT** vocabulary has a higher similarity to the vocabulary of the training data used in this thesis. These results correspond to the findings from Figure 3.2. Although it is feasible to pre-train a model with a custom vocabulary trained using WordPiece tokenization, it is not done in this work as doing so would take a longer time to pre-train than initialising the model with weights from pre-trained models such as **SciBERT** and **BERT$_{\text{BASE}}$** [13, 19].

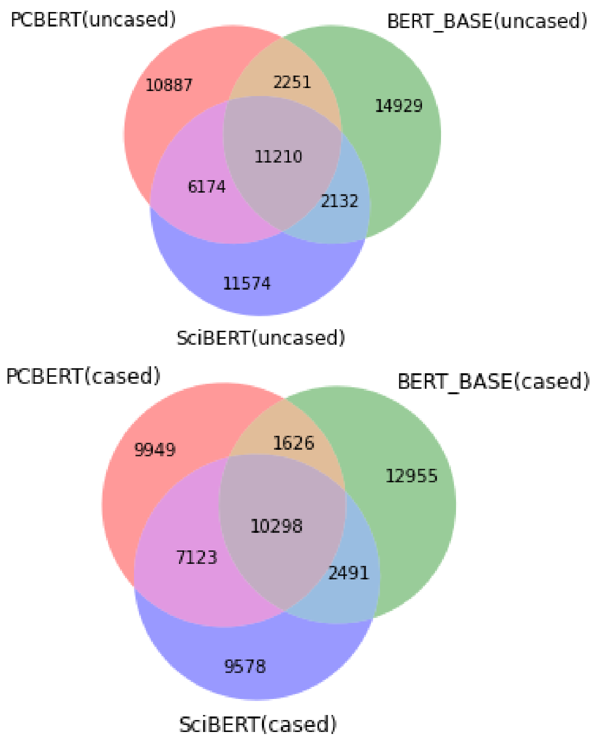


Figure 3.3: Venn diagram of vocabulary trained using WordPiece tokenization with our training corpus (PCBERT) , **SciBERT** and **BERT**$_{\textbf{BASE}}$.

3.1.2 *Computer Science Named Entity Recognition in the Open Research Knowledge Graph (CS-NER)*

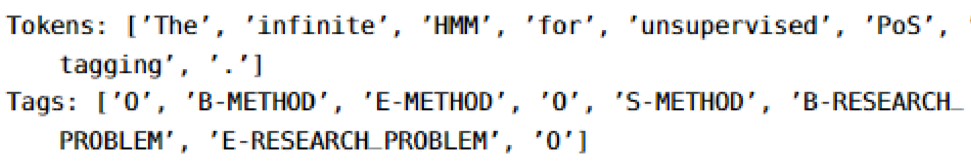
As a way to evaluate the performance of the pre-trained models objectively in the computer science domain, an NER dataset that focuses on a standardized annotation for contribution-centric scholarly entities in the computer science domain is employed. The dataset was proposed by D'Souza and Auer in their work [12]. In their work, they unified annotations from prior work on scholarly domain-specific NER datasets from the computer science domain and standardized the various annotations from the assorted datasets to form seven contribution-centric entities, namely "research problem", "method", "solution", "tool", "resource", "dataset", "language". The field of interest of the datasets studied in [12] ranges from computer linguistics to artificial intelligence as well as computer vision. The entities and the definition according to [12] are given in Table 3.1.

Table 3.1: Entity types and their definitions in CS-NER dataset according to [12].

Entity	Definition
research problem	Theme of the investigation.
method	Exisiting protocols and procedures used to support a solution.
solution	Novel contribution of a work that solves the research problem.
tool	By which means the research problem was solved.
resource	Names of existing data and other references to utilities like the Web, Encyclopedia, etc.
dataset	Name of dataset.
language	The natural language focus of a work. This is mainly found in the computer linguistics branch.

The datasets from previous works that were standardized in [12] were FTD [15], SciERC [33], NLPContributionGraph (NCG) [11], ACL [10] and PapersWithCode (PwC) [41]. All of the datasets used in CS-NER had existing semantic entity types that matched the definition given by [12]. Hence, these were incorporated into the CS-NER dataset. The remaining semantic entities that did not match the contribution-centric entities were discarded. In each of these datasets, texts from research article titles and abstracts were extracted. D'Souza and Auer structured the dataset in such a way that the annotated NER dataset for research titles were separated from the annotated dataset for research abstracts. The titles corpus contained the semantic entities mentioned in Table 3.1, whereas the abstracts corpus only had the entities "research problem" and "method". This discrepancy is due to the fact that all the datasets used in [12] had the same format, where the titles were separated from the abstracts and the abstract corpora had mostly entities that fit into the definition of "research problem" and "method". Both titles and abstracts corpora are split into training, evaluation and testing datasets. For this thesis, only the titles corpus of CS-NER is used for fine-tuning purposes as this corpus has more semantic entities than the abstracts corpus. Furthermore, the abstracts corpus has less records in its datasets than the titles corpus.

The annotation scheme for CS-NER follows an IOBES-scheme. In this scheme, words in texts are treated as tokens and the tokens can be assigned a tag to a specific entity type. A group of tagged tokens make up a chunk and therefore spans of tokens of different lengths can be tagged. The tags of this annotation scheme indicate the relative position and span of the annotated entities and can be broken down into five parts: I-tag stands for a tag inside a chunk, O-tag means no entity type can be assigned to the token, B-tag stands for the beginning of a chunk, E-tag stands for the end of a chunk and S-tag indicates that the chunk is made up of a single token. The abbreviations, except for the O-tag, are prefixed before the entity type, so that both the information about the relative positions of the tag in the chunk as well as the entity type can be expressed. An example is shown in the figure below:



```
Tokens: ['The', 'infinite', 'HMM', 'for', 'unsupervised', 'PoS', '
    tagging', '.']
Tags: ['O', 'B-METHOD', 'E-METHOD', 'O', 'S-METHOD', 'B-RESEARCH_
    PROBLEM', 'E-RESEARCH_PROBLEM', 'O']
```

Figure 3.4: An example record in the CS-NER dataset with IOBES-scheme.

3.1.3 *Workshop on Information Extraction from Scientific Publications (WIESP)*

A thorough search of the relevant literature yielded only one related NER dataset in the physics domain with considerable number of entities. WIESP was a workshop to foster discussion and research using NLP and machine learning [66]. Through this forum, professionals , researchers as well as students can cooperate towards constructing algorithms, models and tools to pave way for machine comprehension of science in the future. The workshop in 2020 presented a dataset that is a derived dataset from the Detecting Entities in the Astrophysics Literature (DEAL) task which is available on their HuggingFace website [65]. This dataset contains text fragments from astrophysics papers with manually tagged astronomical facilities and other entities of interest in the astrophysics domain. Moreover, the task of DEAL was to build a system that is automatic and does not require human intervention that can identify named entities in text fragments from the astrophysics literature [9]. A total of 31 semantic entities were used as labels in this dataset. The entities and their definitions are presented in Table 3.2.

Table 3.2: Entity types and their definitions in WIESP dataset according to [9].

Entity	Definition
Archive	Curated collection of the literature or data. (e.g. MAST)
CelestialObject	Named object in the sky. (i.e. Andromeda galaxy)
CelestialObjectRegion	Named area on/in in a celestial boy. (e.g. Inner galaxy)
CelestialRegion	A defined region projected onto the sky or celestial coordinates. (e.g. GOODS field, l=2, b=15)
Citation	A reference to previous work in the literature. (e.g. Allen et al. 2012)
Collaboration	An organizational entity containing multiple organizations and/or countries. (e.g Plank Collaboration)
ComputingFacility	Facility whose primary purpose is to operate computational resources. (e.g. CINECA supercomputing Centre)
Database	Curated and searchable set of relatable data tables (very similar to Archive). (e.g. Simbad)
Dataset	Curated set of data (Similar to a single data table). (e.g. Gaia EDR3)
EntityOfFutureInterest	A general catch of all entities that may be worth further research in the future.
Event	A conference, workshop or other event that often brings scientists together. (e.g. Protostars and Planets VI)

Table 3.2 (continued)

Entity	Definition
Fellowship	A grant that is focused towards students and/or early career researchers. (e.g. Hubble Fellowship)
Formula	Mathematical formula or equations. (e.g. $F = Gm_1m_2/r^2$)
Grant	Allocation of money and/or time for a research project. (e.g. grant No. 12345)
Identifier	Unique identifier for data, images, etc. (e.g. ALMA 123.12345)
Instrument	A device used to make measurements. (e.g. NIRCam)
Location	A name of location on Earth. (e.g. Canada)
Mission	A spacecraft that is not a telescope or observatory that carries multiple instruments. (e.g. WIND)
Model	Name of scientific or computational model. (e.g. Salpeter IMF)
ObservationalTechniques	Method used to observe celestial objects. (e.g. Resolved Long-slit Spectroscopy)
Observatory	A group of telescopes, that are often similarly located. (e.g. Keck Observatory)
Organization	An organization that is not an observatory. (e.g. NASA)
Person	Name of person. (e.g. A. Einstein)
Proposal	A request for telescope time or funding. (e.g. GN-2014B-Q-26)

Table 3.2 (continued)

Entity	Definition
Software	Name of computer code or language. (e.g. Numpy)
Survey	Organized search of the sky often dedicated to large scale astronomy projects. (e.g. 2MASS)
Tag	A HTML tag. (e.g. <\bold>)
Telescope	Optical instrument to capture images of distant objects. (e.g. Hubble Space Telescope)
TextGarbage	Incorrect text, often multiple punctuations marks with no inner text. (e.g. ',,,,,')
URL	A link to a website. (e.g. https://www.astropy.org/)
Wavelength	A portion of the electromagnetic spectrum (it can be expressed as a particular wavelength, a name or a particular transition). (e.g. 656.46 nm)

The WIESP dataset uses the IOB2-scheme to annotate its text. In contrast to the IOBES-scheme mentioned in 3.1.2, there are no distinctions between the S-tag and the B-tag, that is single tagged tokens receive the B-tag instead of the S-tag. Additionaly, no E-tags are used in this scheme. The IOB2-scheme is similar to the IOB-scheme. The only difference is that every beginning of a token chunk in the IOB2-scheme, including chunks that are made up of single tokens, are tagged with the B-tag. On the other hand, the IOB-scheme tags the beginning of token chunks which consist of a single token with the I-tag.

3.2 METHODOLOGY

Since the goal of this thesis is to learn text representations from the computer science and physics domain, a pre-training procedure is carried out. Subsequently, the models are fine-tuned for the NER task utilizing the datasets shown in Subsection 3.1.2 abd 3.1.3.

3.2.1 *Pre-training*

As discussed in Subsection 2.3.3, the pre-training acomplished in this work is done in a similar manner as with the pre-training of BERT in [13]. In contrast to [13] however, the NSP pre-training is not done for this thesis. One reason for this is because it has been reported in [32] that the performance of BERT is unimproved with NSP. Another reason is because the downstream task in this thesis does not require reasoning about the relationships between pairs of sentences such as the one essential in question answering. Hence, for the pre-training procedure MLM is chosen as the training objective. The objective function for MLM can be expressed with Equation 3.1. The BERT models chosen to initialize the weights before pre-training are **SciBERT** and **BERT$_{\mathbf{BASE}}$**. **SciBERT** was chosen for this thesis because its vocabulary contained more similar words and subwords in its vocabulary as shown in Subsection 3.1.1. **BERT$_{\mathbf{BASE}}$** is also chosen to initiate the weights of the BERT model in this work to investigate whether the performance of the models using the initial weights and vocabulary of **BERT$_{\mathbf{BASE}}$** would suffice on downstream tasks.

$$\max_{\Theta} log\, p_{\Theta}(x|x') \approx \sum_{i=1}^{L} m_i log\, p_{\Theta}(x_i|x_{<i}, x_{>i}) \tag{3.1}$$

$$\mathcal{L}_{CE} = -\sum_{i=1}^{L} m_i log\, p_{\Theta}(x_i|x_{<i}, x_{>i}) \tag{3.2}$$

In Equation 3.1, Θ is the parameters of the model, x and x' indicate the true token sequence and the masked token sequence, respectively. L is the length of the sequence and x_i is the masked token. m_i denotes the presence of a masked token, where $m_i = 1$ denotes that x_i is masked and $m_i = 0$ if otherwise. In order to maximize the objective function, the cross entropy loss is minimized , similar to the approach in 2.1.1. Equation 3.2 shows the cross entropy loss of a single token sequence. To calculate the cross entropy loss of multiple sequences in one batch, the cross entropy loss is averaged over the number of sequences in one batch. To assist in pre-training, BERT models from the HuggingFace [21] platform was used. The models and their respective tokenizers can be downloaded from the website.

To use the models, their respective tokenizers must also be used accordingly. The authors for the various domain-specific BERT models in Section 2.6 used different hyperparameters for pre-training their respective domain-specific BERT models. Table 3.3 shows an overview of the hyperparameters used for pre-training their models including batch size, number of epochs, optimizer settings and sequence length. N/A denotes that no information can be found regarding the hyperparameter. BERT in Table 3.3 refers to **$\text{BERT}_{\text{BASE}}$** and **$\text{BERT}_{\text{LARGE}}$** which were the original BERT models trained by Devlin et al. [13]. **$\text{BERT}_{\text{BASE}}$**, **$\text{BERT}_{\text{LARGE}}$**, **SciBERT** and ClinicalBERT went through two rounds of pre-training, once with a sequence length of 128 and the next round with 512. Additionally, ClinicalBERT trained its first round of pre-training with a batch size of 64 and the second round with a batch size of 8. Pre-training in this thesis was carried out on a NVIDIA A100-PCIE-40GB GPU and the pre-training of each model took 11 to 13 days. An AdamW optimizer was used with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e^{-8}$, weight decay of 0.01 and a learning rate of $2e^{-5}$. Furthermore, a linear decay schedule for the learning rate without a warm-up ratio was used. These hyperparameter settings are similar to the ones used in [19]. A maximum input sequence length of 512 with a batch size of 40 was used. The batch size was set at 40 for the uncased and cased versions of **$\text{BERT}_{\text{BASE}}$** as well as the uncased version of **SciBERT**, whereas a batch size of 30 was used during the pre-training of the cased version of **SciBERT**. The batch sizes were chosen in this manner as this was the maximum size that the GPU was capable to process. Number of epochs for pre-training was set at 5.

Before feeding the text into the models for pre-training, the text are first tokenized by the respective tokenizers of the models into tokens. The tokens created for each sentence are grouped and split into chunks with sequence lengths of 512. This grouping and chunking step is done to concatenate the input sentences and prevent long sentences from getting truncated when it has more tokens than the fixed 512 [21]. This prevents information that might be helpful from being discarded from the pre-training task.

Table 3.3: Hyperparameters used for pre-training different domain-specific BERT models.

Model	**Batch Size**	**Number of Epochs**	**Optimizer Settings**	**Sequence Length**
BERT (**BERT$_{\text{BASE}}$**) (**BERT$_{\text{LARGE}}$**) [13]	256	40	Adam with learning rate = 1×10^{-4}, warm-up steps = 10000, weight decay =0.01, $\beta_1 = 0.9$, $\beta_2 = 0.99$,$\epsilon = 1 \times 10^{-6}$	128, 512
SciBERT [4]	N/A	N/A	N/A	128, 512
BioBERT [28]	192	40	Adam with learning rate = $1e^{-4}$, warm-up steps = 10000, weight decay =0.01, $\beta_1 = 0.9$, $\beta_2 = 0.99$,$\epsilon = 1 \times 10^{-6}$	512
ClinicalBERT [18]	64,8	1	Adam with learning rate = $2e^{-5}$,warm-up steps =10,weight decay=0.01,$\beta_1 = 0.9$,$\beta_2 = 0.999$,$\epsilon = 1 \times 10^{-6}$	128,512
MatSciBERT [16]	256	30	AdamW with learning rate = 1×10^4, warm-up steps = 4.8% of total training steps, weight decay =0.01, $\beta_1 = 0.9$,$\beta_2 = 0.98$,$\epsilon = 1 \times 10^{-6}$	512
MatBERT [61]	192	5	AdamW with learning rate = 5×10^{-5},weight decay =0.01, $\beta_1 = 0.9$, $\beta_2 = 0.999$,$\epsilon = 1 \times 10^{-8}$	N/A
BatteryBERT [19]	256	40	AdamW with learning rate = 5×10^{-5},weight decay =0.01, $\beta_1 = 0.9$,$\beta_2 = 0.999$,$\epsilon = 1 \times 10^{-8}$	512

Language models are usually evaluated on a metric known as perplexity, which is defined in Equation 3.3. s denotes a sentence with tokens $x_1, ..., x_L$ with L being the number of tokens in the sentence. It is a measurement of how confident a language model predicts a test text sample or rather how "surprised" the language model is when it sees new data. Another interpretation of perplexity is as being a branching factor, describing the weighted average number of choices of tokens a language model can have when predicting a subsequent token or word [24]. However, since masked language models are different to the traditional causal language models as described under Section 2.1 that model the probabilities of subsequent words based on preceding words in a sentence, perplexity is not well defined for masked language models [50]. Despite of this, Salazar et al. have introduced the idea of pseudo-perplexity that can be applied to masked language models [50]. In order to score a sentence from a model with pseudo-perplexity, copies of the sentence with each token masked out is created. The special tokens [CLS] and [SEP] are not masked out. The cross entropy for each masked token is summed over copies of the sentences to give a total cross entropy score. The number of tokens, excluding the [CLS] and [SEP] tokens in the sentence is noted. The total cross entropy of the sentence is then divided by the number of tokens in the sentence and exponentiated. Figure 3.5 shows the procedure of calculating the total cross entropy of an example sentence when fed into BERT as a masked language model and Equation 3.4 illustrates the calculation for pseudo-perplexity.

$$Perplexity(s) = e^{-\frac{1}{L}\sum_i^L log\, p_\Theta(x_i|x_{<i})} \tag{3.3}$$

$$Pseudo-perplexity(s) = e^{-\frac{1}{L}\sum_i^L log\, p_\Theta(x_i|x_{<i}, x_{>i})} \tag{3.4}$$

It is also imperative to understand the boundaries of the perplexity metric to evaluate the best and the worst possible perplexities. The best perplexity of a language model is 1, as a language model that is able to predict subsequent or masked words perfectly would assign the token in its vocabulary with a probability of 1 and the log probability is 0, which when exponentiated returns 1. The worst perplexity that a language model can achieve would be the size of its vocabulary. In the worst case, the language model knows absolutely nothing about the data and consequently models the sequence of tokens as a uniform distribution, assigning all tokens that are to be predicted with a probability of $\frac{1}{|V|}$, where $|V|$ is the size of the vocabulary used. As a result, the lower the perplexity, the better the performance of the language model. A proof of the size of vocabulary $|V|$ being the worst score for perplexity is shown in the following equation:

$$\begin{aligned} e^{-\frac{1}{L}\sum_i^L log\, p_\Theta(x_i|x_{<i})} &= e^{-\frac{1}{L}\sum_i^N log\, \frac{1}{|V|}} \\ &= e^{-\frac{1}{N}N log\, \frac{1}{|V|}} \\ &= e^{log|V|} \\ &= |V| \end{aligned} \tag{3.5}$$

Thus, comparing perplexities across models is only sensible, when the models being compared utilize the same vocabulary for its tokenizer. For the evaluation of the masked language models in this thesis, the pseudo-perplexities of the models were computed before and after pre-training on a seperate evaluation corpus extracted from the arXiv text corpus. The evaluation corpus was not used during pre-training and hence the evaluation corpus can be considered as unseen data for the models. To prevent a memory overload during computation of the perplexity scores, only sentences that contain less than 120 tokens are evaluated.

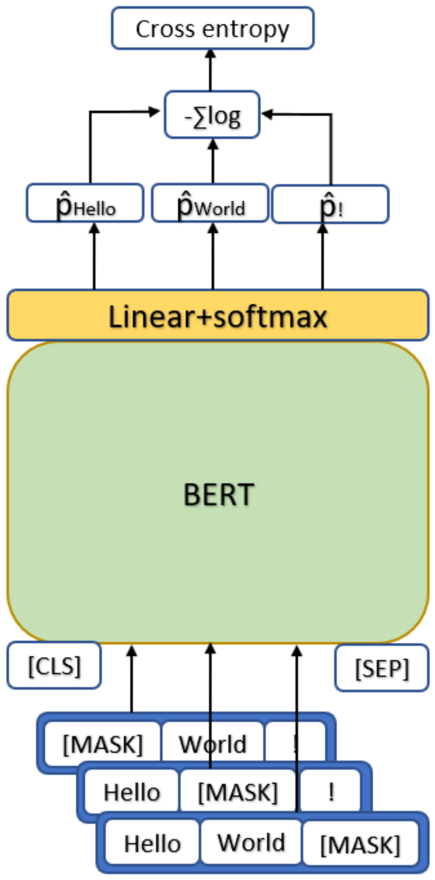


Figure 3.5: Total cross entropy calculated for one sentence as shown in [50].

3.2.2 *Fine-tuning*

After the pre-training stage, the models are subjected to downstream task training with the NER datasets introduced in Subsection 3.1.2 and 3.1.3. Fine-tuning on both NER datasets from CS-NER and WIESP was accomplished by first looking for the best performing hyperparameter settings for each combination of the models together with the datasets. The batch size used is fixed according to dataset used. The batch size for fine-tuning on CS-NER was 40 , whereas the batch size for fine-tuning on WIESP dataset was 10. The disparity of batch sizes between datasets is due to the size of the data input from each dataset. Data input from WIESP was larger as it had longer sequences than the records in CS-NER. The hyperparameters varied in this work are the number of epochs and the learning rates used in the AdamW optimizer. The number of epochs 1 to 8 as well as the learning rate of 1×10^{-4},3×10^{-4},5×10^{-5} or 3×10^{-5} were selected for hyperparameter tuning. This range for the number of epochs was chosen as higher number of epochs would cause longer training durations. In fact, smaller ranges of epochs were used in similar tasks [1]. The range of learning rates matches that of previous works [4, 13, 19].A warm-up ratio of 0.1 with an AdamW optimizer was used for fine-tuning and hyperparameter tuning. For each combination of hyperparameters, the results from 3 different random seeds were used and the average of the evaluation score is calculated. The selected hyperparameters which gave optimum micro F1 score is chosen for the respective model and the dataset the model was evaluated on. The models for each dataset are subsequently trained with their respective optimum hyperparameters with one seed and are assessed using a test dataset. One seed was used as training multiple models on multiple seeds unfortunately requires a lot of disk space to save the models.

Precision, recall and F1 scores are chosen as evaluation metrics for the fine-tuning stage. Precision is defined as a fraction of relevant instances amidst all retrieved instances, while recall is the fraction of retrieved instances among all relevant instances. Precision and recall can be expressed as equations 3.6 and 3.7 respectively. TP is the number of true positives detected, i.e. how many predicted entities were actually correct, FP is the number of false positives, i.e. how many predicted entities were in fact not entities and FN is the number of false negatives, i.e. how many unpredicted entities were actually true entities. The F1-Score is used as an average score of precision and recall.

$$Precision = \frac{TP}{TP + FP} \tag{3.6}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.7}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.8}$$

Calculations for the F1 score for both WIESP and CS-NER are the same. The seqeval metric module from Pytorch was used to compute the mentioned evaluation metrics [39]. However, the definition of TP,FP and FN within the WIESP task differs from the ones used in CS-NER. This is due to the fact, that the WIESP task uses a lenient count for TP, FP and FN, which counts correct entity types with incorrect I/B-prefixes as true positives. Conversely, the IOBES scheme used in CS-NER only allows a strict mode in seqeval and doesn't identify in-correctly prefixed entity labels as true positives, penalising the model not only on entity type but also the relative position of the label.

There also exist two procedures for averaging the evaluation results regarding multilabel tasks. These procedures are coined as micro and macro averages. The micro average of a metric is calculated by summing the individual TP, FP and FN of the system for different sets or labels and apply them to calculate the evaluation metric. Whereas for macro average, the evaluation metric for different labels are calculated and the average of the calculated evaluation metrics of the different labels is taken. While micro averaging favours the majority class, as the resulting performance takes into account the proportion of every class and each observation is given equal weight, macro averaging favours the minority class, as it gives equal weight to each class of the task instead. Since micro averaging takes into account the imbalance of labels in the dataset, it is often used in NER [28, 57]. Hence, the micro averaged F1 metrics are used in this work to evaluate the models. A detailed illustration of the equations for calculating micro precision and micro recall is depicted in Equations 3.9 and 3.10 , respectively. Here, $|L|$ denote the number of labels and in the case of NER denote the number of entity types. The calculation for micro and macro F1 scores is shown in Equations 3.11 and 3.12, respectively.

$$Precision_{micro} = \frac{\sum_{i=1}^{|L|} TP_i}{\sum_{i=1}^{|L|} TP_i + \sum_{i=1}^{|L|} FP_i} \tag{3.9}$$

$$Recall_{micro} = \frac{\sum_{i=1}^{|L|} TP_i}{\sum_{i=1}^{|L|} TP_i + \sum_{i=1}^{|L|} FN_i} \tag{3.10}$$

$$F1_{micro} = 2 * \frac{Precision_{micro} * Recall_{micro}}{Precision_{micro} + Recall_{micro}} \tag{3.11}$$

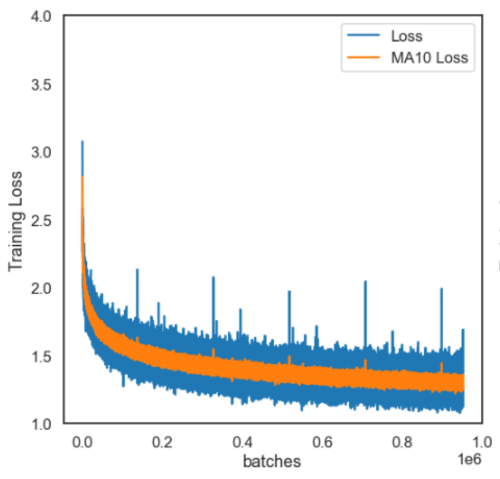
$$F1_{macro} = \frac{1}{|L|} \sum_{i=1}^{|L|} F1_i \tag{3.12}$$

4 RESULTS AND DISCUSSION

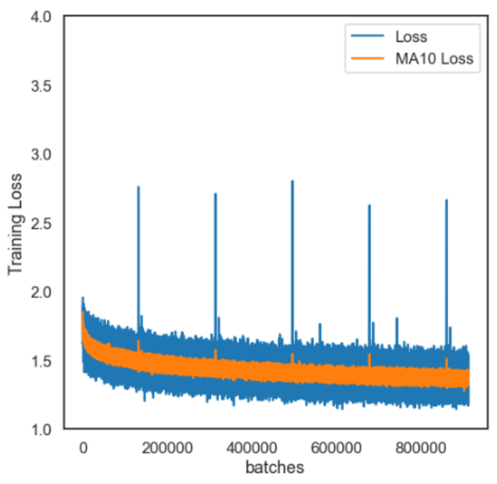
In this chapter, the results from the experiments carried out in Chapter 3 are presented and discussed with regard to previous works and research. Firstly, the results from pre-training the different BERT models are presented and compared to previous performances achieved by previous works. Finally, the results on the fine-tuning performances of the models pre-trained in this work on the NER-tasks are introduced and discussed.

4.1 PRE-TRAINING

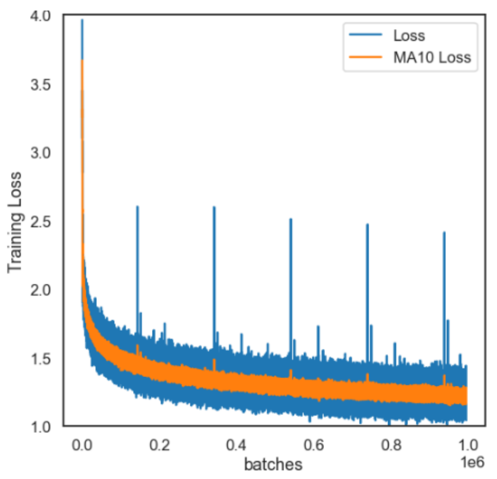
The cross-entropy loss curves for both the uncased and cased variants of **BERT$_{\textbf{BASE}}$** as well as **SciBERT** were recorded throughout pre-training. Figure 4.1 shows the cross-entropy loss curves that were recorded during the pre-training stages of the models. The term "uncased" in Figure 4.1 signifies that an uncased vocabulary of the model was used, whereas the term "cased" stands for the usage of the cased vocabulary. From Figure 4.1, it is clear that pre-training the model with initialized weights from **SciBERT** has a lower initial cross-entropy loss (1.8 for uncased model,1.99 for cased model) when compared to the model initialized with weights from the **BERT$_{\textbf{BASE}}$** model (3,07 for uncased model,3.96 for cased model). This behaviour is similar to the results found in [19]. A popular explanation for this observation is that **SciBERT** was initialized with weights from the **BERT$_{\textbf{BASE}}$** model for its pre-training and pre-trained on scientific texts [4]. Hence, the **SciBERT** language model has improved the weights from the **BERT$_{\textbf{BASE}}$** model and can predict masked tokens better than **BERT$_{\textbf{BASE}}$**. Although this is generally accepted, it does not directly indicate that a language model with lower cross-entropy loss in masked language modelling outperforms a model with higher loss on downstream tasks. This was especially noticeable during the question-answering task in the field of battery research used in [19] where the model pre-trained on weights from cased **BERT$_{\textbf{BASE}}$** outperformed that pre-trained with cased **SciBERT** weights. Overall, the cross-entropy loss decreases continuously and the models that stop pre-training after 5 epochs demonstrated good performances on the NER tasks as will be shown later.



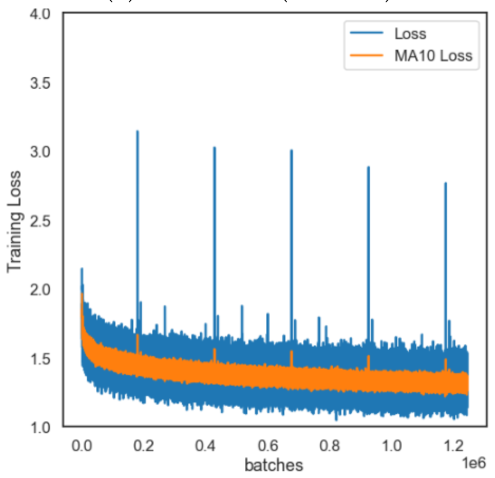
(a) PCBERT(uncased)



(b) PCSciBERT(uncased)



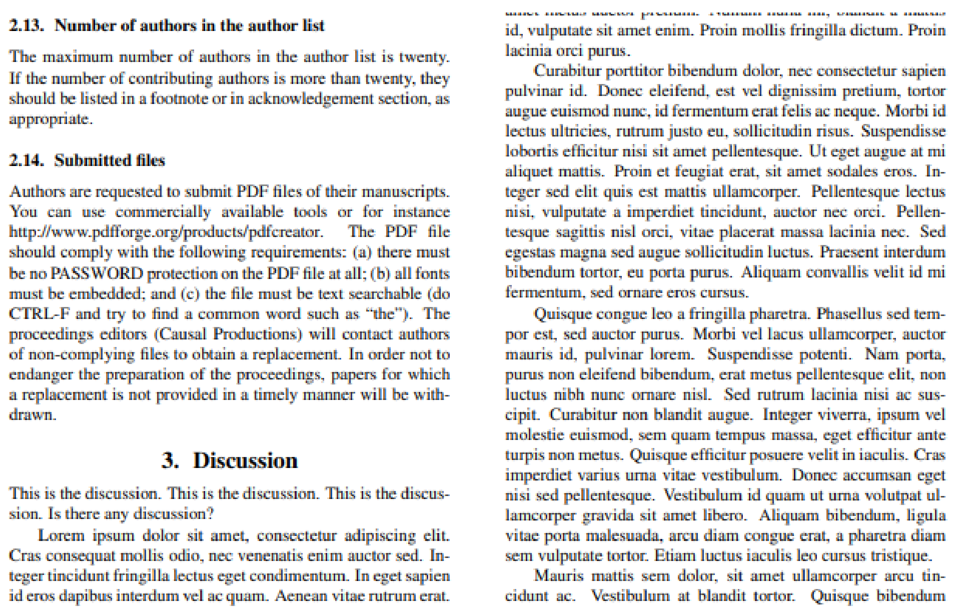
(c) PCBERT(cased)



(d) PCSciBERT(cased)

Figure 4.1: Cross-entropy loss recorded during pre-training of models in this thesis. The MA10 Loss describes the cross-entropy loss averaged over 10 steps.

What is also interesting to note are the five spikes in the pre-training loss curves in Figure 4.1. Thorough analysis of the training data for pre-training shows that there are also extracted texts from published research articles on the official arXiv repository that contain placeholder sentences. An example text fragment from a research article in the computer science domain of arXiv containing the placeholder texts is shown in Figure 4.2. In addition, there are also hidden texts in the PDF-files that could be extracted with the PyMuPDF library that were present in research papers as found in [23]. It is assumed that the authors of these publications had used these texts as a placeholder during the writing of these articles and have not updated the publications to a more formal version ever since. It is found that the training batch with the highest proportion of said placeholder text corresponded to the batches which caused the spikes in the pre-training loss curves shown in 4.1. Due to the fact that the cased and uncased versions of both **BERT$_{\text{BASE}}$** and **SciBERT** were primarily pre-trained on texts from the English language, this may explain why the model was not able to correctly predict the masked tokens during training. Furthermore, the placeholder texts used in these research articles are known as Lorem Ipsum texts and do not make sense semantically. Although these texts exist in the training dataset, the number of research articles containing these texts were negligible. There are current Python libraries such as langdetect [73], which is a Python version of Google's Java language-detection library [72], which can detect the type of languages in a text but they cannot detect domain-specific jargon. Hence, using these libraries to screen out unknown terms would also screen out the scientific terms needed to pre-train the model. Our findings also show that out models have learned to model these placeholder texts as can be seen from the decrease in its cross-entropy loss, though they are not relevant to the domains of interest in this work.



2.13. Number of authors in the author list

The maximum number of authors in the author list is twenty. If the number of contributing authors is more than twenty, they should be listed in a footnote or in acknowledgement section, as appropriate.

2.14. Submitted files

Authors are requested to submit PDF files of their manuscripts. You can use commercially available tools or for instance http://www.pdfforge.org/products/pdfcreator. The PDF file should comply with the following requirements: (a) there must be no PASSWORD protection on the PDF file at all; (b) all fonts must be embedded; and (c) the file must be text searchable (do CTRL-F and try to find a common word such as "the"). The proceedings editors (Causal Productions) will contact authors of non-complying files to obtain a replacement. In order not to endanger the preparation of the proceedings, papers for which a replacement is not provided in a timely manner will be withdrawn.

3. Discussion

This is the discussion. This is the discussion. This is the discussion. Is there any discussion?

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras consequat mollis odio, nec venenatis enim auctor sed. Integer tincidunt fringilla lectus eget condimentum. In eget sapien id eros dapibus interdum vel ac quam. Aenean vitae rutrum erat. [illegible] id, vulputate sit amet enim. Proin mollis fringilla dictum. Proin lacinia orci purus.

Curabitur porttitor bibendum dolor, nec consectetur sapien pulvinar id. Donec eleifend, est vel dignissim pretium, tortor augue euismod nunc, id fermentum erat felis ac neque. Morbi id lectus ultricies, rutrum justo eu, sollicitudin risus. Suspendisse lobortis efficitur nisi sit amet pellentesque. Ut eget augue at mi aliquet mattis. Proin et feugiat erat, sit amet sodales eros. Integer sed elit quis est mattis ullamcorper. Pellentesque lectus nisi, vulputate a imperdiet tincidunt, auctor nec orci. Pellentesque sagittis nisl orci, vitae placerat massa lacinia nec. Sed egestas magna sed augue sollicitudin luctus. Praesent interdum bibendum tortor, eu porta purus. Aliquam convallis velit id mi fermentum, sed ornare eros cursus.

Quisque congue leo a fringilla pharetra. Phasellus sed tempor est, sed auctor purus. Morbi vel lacus ullamcorper, auctor mauris id, pulvinar lorem. Suspendisse potenti. Nam porta, purus non eleifend bibendum, erat metus pellentesque elit, non luctus nibh nunc ornare nisl. Sed rutrum lacinia nisi ac suscipit. Curabitur non blandit augue. Integer viverra, ipsum vel molestie euismod, sem quam tempus massa, eget efficitur ante turpis non metus. Quisque efficitur posuere velit in iaculis. Cras imperdiet varius urna vitae vestibulum. Donec accumsan eget nisi sed pellentesque. Vestibulum id quam ut urna volutpat ullamcorper gravida sit amet libero. Aliquam bibendum, ligula vitae porta malesuada, arcu diam congue erat, a pharetra diam sem vulputate tortor. Etiam luctus iaculis leo cursus tristique.

Mauris mattis sem dolor, sit amet ullamcorper arcu tincidunt ac. Vestibulum at blandit tortor. Quisque bibendum

Figure 4.2: Lorem Ipsum sentences found under the "Discussion" section from [63].

The pseudo-perplexities of the models before and after pre-training are shown in Table 4.1. From Table 4.1, it can concluded that pre-training on texts that are rich in computer science and physics texts have led to an increase in performance for the BERT models in modelling texts in these domains. Although a comparison of the pseudo-perplexities should only be compared within models using the same vocabularies by their respective tokenizers, it is worth noting that all models achieved pseudo-perplexities of just above 3.

Table 4.1: Pseudo-perplexities of BERT models before and after pre-training on physics and computer science training corpus.

Models	**Pseudo-perplexity**	
	Before Pre-training	**After Pre-training**
BERT$_{\textbf{BASE}}$(cased)	8.26	3.15
BERT$_{\textbf{BASE}}$(uncased)	10.04	3.40
SciBERT(cased)	7.95	3.46
SciBERT(uncased)	5.28	3.68

4.2 FINE-TUNING

Table 4.2 shows the values of the F1 scores for each entity type and for each model. The procedure of calculating the F1 scores for each entity type is the same by using Equation 3.8. The values in parenthesis next to the F1 scores are the absolute difference between the F1 score of the model, which was pre-trained in this thesis, and the F1 score of the initial model, from which the model was initialised from e.g. difference between F1 score of PCBERT(cased) and F1 score of cased version of **BERT$_{\textbf{BASE}}$**, difference between F1 score of PCSciBERT(cased) and F1 score of cased version of SciBERT etc. The values of F1 scores reported in this work are all in percentages, unless stated otherwise. The total annotated entities for each entity type is also presented in Table 4.2. The numbers do not reflect the unique set of annotated entities. Red cells in Table 4.2 indicate the lowest F1 scores achieved by the model, whereas green cells indicate the highest F1 scores from the model. Values in bold represent the highest absolute increase of F1 scores and underlined values show the second highest absolute increase of the F1 scores.

From Table 4.2, it is clear that the entity type "language" obtained the highest F1 scores across all models. This corresponds to the results observed in [12], as the human annotators and authors who curated the dataset had the highest consensus score for this entity and hence it is easier for the model to detect this entity type. All models performed worst on the entity type "method". This also agrees with the findings of the authors in [12] due to the frequent confusion among the annotators when deciding on whether the entity belongs to the entity type "method" or "tool". As a consequence, the "tool" entity type received the second lowest F1 scores from all models. The largest improvement for the F1 score comes from PCBERT(cased) with an improvement of 4.1% for the entity type "dataset". The models pre-trained on the arXiv text corpus in this thesis also caused F1 scores for some entities to deteriorate with the highest reduction of F1 score obtained from PCBERT(uncased) for the entity type "language" with a F1 score reduction of 3.87%. This could hint that pre-training a language model on domain specific text data could worsen its performance on downstream tasks that require more general knowledge of the language. Since the entity type "language" defined by [12] focuses on the natural languages, this entity type does not require the domain specific knowledge learned in the physics or computer science text corpus. Further analysis also shows that all the models pre-trained in this thesis improved the F1 scores for at least 5 out of 7 entity types. The entity types which were improved by all models are "research problem", "resource" and "tool", which are common entities that can be found from research articles.

Table 4.2: F1 scores from models pre-trained in this thesis according to entity types for CS-NER dataset. Values without parenthesis denote the F1 score and values in parenthesis denote absolute difference from initial model. Green cells indicate the highest F1 score from the model and red cells indicate the lowest F1 score from the model. Values in bold indicate the highest improvements in F1 score from the model compared to its original before pre-training and underlined values denote the second highest improvements in F1 score.

Entity	**Models**			
	PCBERT(cased)	**PCBERT(uncased)**	**PCSciBERT(cased)**	**PCSciCBERT(uncased)**
dataset (228)	**73.31 (4.1)**	70.61 (-1.01)	68.35 (-0.98)	70.87 (-0.99)
language (499)	86.57 (0.33)	84.76 (-3.87)	89.41 (0.65)	**88.69 (1.88)**
method (2768)	58.78 (3.53)	57.73 (0.18)	55.53 (-0.44)	55.65 (1.62)
research problem (4070)	72.74 (0.17)	73.38 (0.81)	73.68 (0.31)	74.17 (0.36)
resource (3226)	76.48 (0.4)	77.09 (0.4)	**76.83 (0.83)**	76.65 (0.26)
solution (8316)	81.81 (-0.3)	**81.93 (1.1)**	82.55 (0.78)	81.81 (-0.36)
tool (743)	65.5 (0.45)	64.81 (0.11)	65.59 (0.76)	66.46 (1.43)

The overall micro F1 scores for each model when tested on CS-NER dataset is recorded in Table 4.3. Results from Table 4.3 show that for all models pre-trained in this work, the micro F1 scores for the CS-NER dataset have been improved. Comparing the models pre-trained in this thesis and their initial models, with which their weights were initialized with before pre-training, PCBERT(uncased) and PCSciBERT(cased) both improved their corresponding initial models by 0.69%, the largest improvement among all the models. The largest micro F1 score achieved during fine-tuning on the CS-NER dataset came from PCSciBERT(cased) with a micro F1 score of 76.22%. This shows that SciVOCAB used by SciBERT does help in achieving higher performance for downstream tasks of scientific domains. Further analysis also indicate that all the cased versions of the models used during fine-tuning with the exception of PCBERT achieved higher micro F1 scores than their uncased counterparts. This finding supports the notion that the cased variants do improve the models capability of identifying and classifying tokens to the correct entity type, although the increase in micro F1 scores for all models are less than 1%. Considering that most entities in the CS-NER dataset contain words that are capitalized, this result is to be expected. Authors in [28] also came to the conclusion, that models with the cased vocabulary achieve higher performances for NER tasks.

Table 4.3: Micro F1 scores of models fine-tuned on CS-NER dataset.

Models	**Micro F1**
BERT$_{\textbf{BASE}}$(cased)	74.88
BERT$_{\textbf{BASE}}$(uncased)	74.76
SciBERT(cased)	75.53
SciBERT(uncased)	75.41
PCBERT(cased)	75.32
PCBERT(uncased)	75.45
PCSciBERT(cased)	76.22
PCSciBERT(uncased)	75.67

Similar to Table 4.2, the F1 results for each entity type in the WIESP dataset are presented in Table 4.4. The coloured cells as well as the bolded and underlined values in Table 4.4 denote similar meaning as explained for Table 4.2. All models fine-tuned in this thesis achieved the highest F1 scores for the entity type "Fellowship". To the contrary, all of the models obtained the lowest F1 scores for the entity type "Instrument". Since neither "Fellowship" nor "Instrument" have the highest or lowest counts of annotated entities in the test dataset, the number of annotated entities in the dataset does not seem to be the reason that the F1 scores for these entity types lie on different extremes. The models pre-trained in this thesis also achieved higher F1 scores for most of the entity types compared to the models, whose weights were used to initialize for pre-training. Both PCBERT(cased) and PCSciBERT(cased) accomplished the best improvement in F1 scores for the "Software" entity within the test dataset. Moreover, the "Model" entity also received the highest increase in F1 score from PCBERT(uncased) and the second highest improvements from PCBERT(cased) and PCSciBERT(cased). Both these entities relate closely to computer science and physic domains. This could be a possible indication that pre-training the models on texts in these domains led to a better understanding for the domain specific language and hence lead to better identification of these domain specific entities.

Table 4.4: F1 scores from models pre-trained in this thesis according to entity types for WIESP dataset. Values without parenthesis denote the F1 score and values in parenthesis denote absolute difference from initial model. Green cells indicate the highest F1 score from the model and red cells indicate the lowest F1 score from the model. Bolded values indicate the highest improvements in F1 score from the model compared to its original before pre-training and underlined values denote the second highest improvements in F1 score.

Entity	**Models**			
	PCBERT (cased)	**PCBERT (uncased)**	**PCSciBERT (cased)**	**PCSciCBERT (uncased)**
Archive (359)	84.35 (4.88)	84.19 (3.75)	83.84 (1.72)	84.54 (1.88)
CelestialObject (3609)	63.09 (1.05)	61.73 (0.85)	63.46 (1.55)	63.04 (0.14)
CelestialObjectRegion (723)	44.66 (5.04)	54.2 (1.95)	44.13 (8.15)	**68.59 (34.94)**
CelestialRegion (209)	78.99 (6.67)	78.03 (5.39)	79.24 (4.26)	78.79 (2.55)
Citation (8621)	90.67 (2.0)	90.18 (0.44)	90.36 (1.67)	89.92 (-0.13)
Collaboration (428)	92.37 (1.44)	92.68 (0.42)	92.97 (0.74)	92.64 (0.44)

Table 4.4 (continued)

Entity	**Models**			
	PCBERT (cased)	**PCBERT (uncased)**	**PCSciBERT (cased)**	**PCSciCBERT (uncased)**
ComputingFacility (607)	64.49 (2.69)	64.45 (-1.25)	71.13 (3.02)	68.03 (-0.64)
Database (342)	88.54 (6.65)	88.75 (2.97)	88.76 (4.56)	88.57 (2.23)
Dataset (516)	80.31 (7.68)	80.33 (4.85)	80.5 (2.95)	80.49 (2.49)
EntityOfFutureInterest (435)	56.38 (2.77)	57.44 (1.46)	56.94 (-0.01)	60.62 (2.41)
Event (59)	60.78 (4.09)	61.02 (1.31)	62.12 (0.48)	61.52 (-0.08)
Fellowship (607)	96.72 (3.02)	96.43 (0.29)	97.3 (0.39)	96.84 (0.23)
Formula (3452)	86.47 (3.09)	87.35 (0.51)	87.51 (0.19)	87.54 (0.07)
Grant (5259)	49.18 (6.24)	45.07 (6.74)	49.08 (4.29)	49.87 (7.16)
Identifier (180)	74.6 (8.67)	74.2 (6.45)	71.46 (4.06)	73.95 (-1.08)
Instrument (1102)	3.37 (2.5)	0.0 (-0.85)	0.0 (-0.38)	1.15 (0.38)
Location (2256)	63.58 (8.47)	61.04 (4.68)	62.41 (3.52)	63.47 (2.5)
Mission (204)	89.08 (4.62)	89.51 (1.65)	90.14 (0.92)	89.37 (0.24)
Model (3208)	68.91 (9.02)	**69.77 (12.51)**	72.73 (12.08)	71.98 (5.94)
ObservationalTechniques (91)	79.32 (6.46)	75.02 (5.17)	76.83 (2.02)	76.13 (1.11)
Observatory (1326)	92.86 (6.62)	93.79 (1.15)	93.61 (0.86)	93.25 (0.49)
Organization (11399)	77.22 (3.08)	78.29 (2.67)	78.91 (0.74)	77.91 (1.7)
Person (6085)	84.25 (6.89)	83.14 (4.28)	84.19 (4.17)	83.76 (2.22)
Proposal (180)	84.74 (5.93)	80.64 (1.25)	80.51 (-2.53)	80.67 (-1.42)
Software (1386)	**53.32 (23.5)**	51.89 (5.55)	**53.82 (15.32)**	57.3 (6.54)
Survey (1188)	66.67 (1.82)	67.88 (0.34)	69.61 (0.91)	70.46 (3.27)
Tag (115)	77.93 (5.12)	77.34 (2.55)	77.11 (1.65)	76.98 (0.8)
Telescope (2346)	19.95 (0.53)	24.22 (2.99)	20.43 (2.63)	23.92 (-1.12)

Table 4.4 (continued)

Entity	**Models**			
	PCBERT (cased)	**PCBERT (uncased)**	**PCSciBERT (cased)**	**PCSciCBERT (uncased)**
TextGarbage (283)	52.44 (6.84)	52.07 (3.77)	52.06 (4.22)	51.09 (3.57)
URL (405)	90.67 (1.85)	89.73 (1.06)	89.8 (0.6)	90.52 (-0.14)
Wavelength (4643)	87.22 (1.4)	86.01 (1.06)	86.24 (-0.25)	86.2 (0.14)

It is worth discussing the interesting findings revealed by the results of Table 4.5, which shows the overall micro F1 scores of the models fine-tuned on WIESP dataset. Both PCBERT(cased) and PCBERT-(uncased) improved their original models used for initiation in pre-training by 3.85% and 1.92%, respectively. Not to mention, both these models obtained a higher improvement than their PCSciBERT counterparts, a similar finding observed on the CS-NER dataset. This suggests that the SciBERT models, that have been pre-trained on scientific texts and have their own scientific vocabulary, can't be improved as much as their **BERT**$_{\textbf{BASE}}$ counterparts, as SciBERT has already obtained much more domain specific comprehension. PCSciBERT(cased) and PCSciBERT(uncased) improved their original models by 1.49% and 0.8%, respectively.

Table 4.5: Micro F1 scores of models fine-tuned on WIESP dataset.

Models	**Micro F1**
BERT$_{\textbf{BASE}}$(cased)	77.46
BERT$_{\textbf{BASE}}$(uncased)	79.12
SciBERT(cased)	80.7
SciBERT(uncased)	80.74
PCBERT(cased)	81.31
PCBERT(uncased)	81.04
PCSciBERT(cased)	82.19
PCSciBERT(uncased)	81.54

Although PCBERT(cased) and PCBERT(uncased) achieved higher improvements than their PCSciBERT counterparts, both PCSciBERT-(cased) and PCSciBERT(uncased) achieved the highest and second highest micro F1 scores among all the models when fine-tuning on WIESP dataset, respectively. Similar to the findings from the CS-NER dataset as shown in Table 4.3, it can be inferred that the application of SciVOCAB in the BERT language model allows the model to comprehend the vocabulary used in computer science and physics texts, which mostly already exist in the scientific domain of SciVOCAB. Furthermore, PCSciBERT(cased) achieved higher micro F1 scores than PCSciBERT(uncased) in both datasets. This implies that the cased versions do play a role in improving the performance in NER tasks, especially when most of the entities begin with capital letters. The improvements in micro F1 scores from the CS-NER dataset is smaller than the improvements achieved in the WIESP dataset. This could be caused by the lower proportion of computer science texts in the text corpus as mentioned in Subsection 3.1.1. Besides that, a strict evaluation mode was used when evaluating the F1 results on CS-NER dataset. The masked language models PCSciBERT(cased) and PCSciBERT(uncased) have been deposited on `https://huggingface.co/jmzk96/PCSciBERT_cased` and `https://huggingface.co/jmzk96/PCSciBERT_uncased`, respectively. Since both of these models achieved the best and second best performances for both NER datasets, they may be useful for further applications in future work and can be easily accessed with the Python transformers library [47].

Fine-tuning the pre-trained models with the CRF layer also gave similar results to the models that were just fine-tuned with a linear layer on top of the BERT embeddings. Table 4.6 shows the F1 scores of the models with CRF layer according to entity types fine-tuned on CS-NER dataset. Albeit the values of F1 scores are lower than the models with just a linear layer on top of the BERT embeddings, they show that pre-training the BERT models on computer science and physics texts do lead to an improvement in the domain knowledge of the embeddings, irrespective of the final output layer in the model for the downstream tasks. Analogous to the results obtained by models without the CRF layer, the "language" entity type received the highest F1 score, whereas the entity type "method" received the lowest.

Table 4.6: F1 scores from BERT+CRF models according to entity types for CS-NER dataset. Values without parenthesis denote the F1 score and values in parenthesis denote absolute difference from initial model. Green cells indicate the highest F1 score from the model and red cells indicate the lowest F1 score from the model. Bolded values indicate the highest improvements in F1 score from the model compared to its original before pre-training and underlined values denote the second highest improvements in F1 score.

Entity	**Models**			
	PCBERT (cased)+CRF	**PCBERT (uncased)+CRF**	**PCSciBERT (cased)+CRF**	**PCSciCBERT (uncased)+CRF**
dataset (228)	67.81 (-1.29)	67.55 (-2.66)	67.23 (0.15)	66.53 (-1.56)
language (499)	🟩 83.48 (-2.23)	🟩 84.62 (-3.6)	🟩 88.02 (2.88)	🟩 82.42 (-4.38)
method (2768)	🟥 **42.68 (4.31)**	🟥 43.52 (-0.57)	🟥 41.95 (-1.85)	🟥 44.64 (-0.52)
research problem (4070)	68.29 (-0.2)	**69.86 (0.75)**	70.12 (2.04)	69.5 (1.25)
resource (3226)	73.69 (1.49)	74.5 (0.14)	74.69 (0.91)	72.8 (1.31)
solution (8316)	76.95 (-1.64)	79.04 (0.67)	78.0 (2.24)	**78.2 (3.14)**
tool (743)	60.77 (-0.97)	59.73 (-0.48)	**59.37 (4.99)**	61.03 (0.81)

Similar to the results shown in Table 4.4, the models pre-trained on the computer science and physics texts also improved the F1 scores on the WIESP dataset for most entity types even as the CRF layer was added. However, in contrast to the CS-NER dataset, different entity types received the highest and lowest F1 scores. This implies that the different types of output layers of the model do change the outcome of the performance from the models and the performance is not dependent on the datasets labels, which was the case in the CS-NER dataset. Both PCBERT(uncased)+CRF as well as from PCSciBERT(uncased)+CRF provided the highest improvements in F1 scores for the entity type "Mission", which is highly related to the astrophysics domain. Ambiguous and less related entities such as "EntityOfFutureInterest" and "TextGarbage" received the lowest F1 scores. This could indicate that the BERT embeddings pre-trained in this work focus more on domain specific vocabulary.

Table 4.7: F1 scores from BERT+CRF models according to entity types for WIESP dataset. Values without parenthesis denote the F1 score and values in parenthesis denote absolute difference from initial model. Green cells indicate the highest F1 score from the model and red cells indicate the lowest F1 score from the model. Bolded values indicate the highest improvements in F1 score from the model compared to its original before pre-training and underlined values denote the second highest improvements in F1 score.

Entity	**Models**			
	PCBERT (cased)+CRF	**PCBERT (uncased)+CRF**	**PCSciBERT (cased)+CRF**	**PCSciCBERT (uncased)+CRF**
Archive (359)	85.74 (2.52)	87.79 (1.71)	89.8 (2.85)	88.03 (1.86)
CelestialObject (3609)	84.43 (9.8)	84.43 (8.69)	84.89 (5.23)	83.72 (4.14)
CelestialObjectRegion (723)	17.01 (3.14)	18.22 (-0.4)	20.46 (1.75)	17.5 (4.07)
CelestialRegion (209)	**49.26 (17.31)**	41.81 (11.95)	53.35 (16.4)	54.37 (5.07)
Citation (8621)	95.27 (5.5)	95.07 (1.02)	95.92 (1.01)	94.36 (-0.54)
Collaboration (428)	70.91 (2.88)	75.08 (-1.82)	80.19 (-1.55)	77.0 (1.26)
ComputingFacility (607)	68.19 (1.29)	70.44 (2.25)	71.82 (5.82)	68.37 (1.06)
Database (342)	75.74 (0.82)	75.2 (1.58)	74.88 (2.58)	74.38 (2.1)

Table 4.7 (continued)

Entity	**Models**			
	PCBERT (cased)+CRF	**PCBERT (uncased)+CRF**	**PCSciBERT (cased)+CRF**	**PCSciCBERT (uncased)+CRF**
Dataset (516)	46.2 (6.74)	46.63 (8.13)	52.59 (10.76)	48.51 (1.75)
EntityOfFutureInterest (435)	0.0 (0.0)	0.59 (-1.35)	2.74 (-2.01)	1.16 (-1.67)
Event (59)	49.09 (-1.76)	60.61 (7.83)	**57.26 (19.99)**	54.63 (7.84)
Fellowship (607)	63.04 (1.97)	63.3 (-1.51)	65.72 (4.84)	62.63 (1.72)
Formula (3452)	64.03 (15.1)	66.42 (4.07)	65.65 (-0.52)	65.99 (1.07)
Grant (5259)	61.71 (1.51)	61.47 (1.0)	61.95 (2.86)	61.32 (0.95)
Identifier (180)	72.77 (0.86)	70.56 (0.49)	73.99 (3.93)	73.2 (-1.55)
Instrument (1102)	66.57 (9.63)	68.38 (14.03)	71.02 (7.87)	72.92 (9.99)
Location (2256)	90.96 (2.86)	90.5 (1.27)	91.05 (2.08)	90.29 (0.96)
Mission (204)	32.27 (17.1)	**42.08 (24.78)**	45.6 (12.07)	**45.05 (14.0)**
Model (3208)	57.11 (9.23)	54.6 (7.28)	57.0 (7.39)	55.51 (5.16)
ObservationalTechniques (91)	2.33 (-13.05)	14.75 (4.31)	23.91 (7.74)	22.36 (-0.04)
Observatory (1326)	81.6 (3.61)	80.18 (2.64)	82.35 (0.97)	82.19 (2.26)
Organization (11399)	89.26 (3.56)	89.39 (1.55)	90.01 (2.31)	88.72 (1.52)
Person (6085)	96.26 (1.44)	96.14 (1.15)	96.61 (2.66)	96.28 (1.74)
Proposal (180)	44.48 (4.75)	45.52 (3.38)	43.71 (-8.29)	50.61 (-4.21)
Software (1386)	78.06 (8.62)	76.83 (5.55)	77.4 (4.07)	76.39 (3.98)
Survey (1188)	75.63 (7.83)	76.72 (8.35)	77.73 (8.14)	77.79 (5.03)
Tag (115)	82.41 (-0.77)	86.17 (2.69)	84.15 (0.86)	86.91 (5.3)
Telescope (2346)	77.11 (7.55)	77.2 (5.1)	78.48 (5.99)	78.65 (4.31)
TextGarbage (283)	0.0 (0.0)	0.0 (-0.46)	2.3 (-0.85)	3.15 (0.07)
URL (405)	98.12 (0.95)	97.81 (0.22)	98.15 (0.35)	98.19 (0.3)

Table 4.7 (continued)

Entity	**Models**			
	PCBERT (cased)+CRF	**PCBERT (uncased)+CRF**	**PCSciBERT (cased)+CRF**	**PCSciCBERT (uncased)+CRF**
Wavelength (4643)	79.49 (5.32)	80.26 (3.82)	81.99 (3.96)	81.25 (1.86)

Although the BERT models pre-trained in this work do generally improve the F1 scores for both CS-NER and WIESP datasets, the models with the CRF layer as the output layer performed worse than their counterpart models that do not have the CRF layer. Tables 4.8 and 4.9 show the micro F1 scores of the pre-trained models in this work fine-tuned with a CRF layer on the CS-NER and WIESP datasets, respectively. The micro F1 scores from the CS-NER suffered a higher decline in micro F1 scores when compared to the WIESP dataset. It is speculated that this might be due to the evaluation method of seqeval module in Pytorch when evaluating the micro F1 scores on labels using the IOBES-scheme. Since the seqeval evaluates the output in an IOBES-scheme on a strict mode, it penalizes labels that have the wrong IOBES-prefix even though the correct entity type has been predicted.

Table 4.8: Micro F1 scores of BERT models pre-trained in this work with CRF layer fine-tuned on CS-NER dataset.

Models	**Micro F1**
PCBERT(cased)+CRF	69.57
PCBERT(uncased)+CRF	71.29
PCSciBERT(cased)+CRF	71.46
PCSciBERT(uncased)+CRF	70.74

Further inspection of the transition scores of the CRF layer is done to investigate the cause of the lower performances from the models. The transition scores were obtained from the CRF layer and subsequently normalised with a softmax function. Since the dimensions of the transition matrix are $\mathbb{R}^{k \times k}$, where k is the number of label tags depending on dataset, the transition scores can be visualized as a heatmap. Figures 4.3 and 4.4 show the transition scores of the best performing pre-trained BERT model with CRF layer for the CS-NER and WIESP datasets, respectively. For both CS-NER and WIESP datasets, PC-SciBERT(cased)+CRF achieved the highest micro F1 scores among the

Table 4.9: Micro F1 scores of BERT models pre-trained in this work with CRF layer fine-tuned on WIESP dataset.

Models	Micro F1
PCBERT(cased)+CRF	79.74
PCBERT(uncased)+CRF	79.82
PCSciBERT(cased)+CRF	81.76
PCSciBERT(uncased)+CRF	80.43

models with the added CRF layer. The label tags on the y-axis in Figures 4.3 and 4.4 represents the label tag of a current arbitrary position in a sequence of tag predictions and the label tags on the x-axis represent the label tag of the next position in the sequence of tag predictions. Hence, the transition scores presented in the heatmaps can be interpreted as the probability of label tag on the x-axis appearing after the label tag on the y-axis. Analysis of the transition scores from the CRF layer for the CS-NER dataset found evidence that the CRF layer scarcely models the transition between prediction tags. The transition of I-tags to I-tags for most entity types including "language", "method", "research problem" and "resource" is learned by the CRF layer, which is shown by the high transition scores between I-tags of the previously mentioned entity types. The transition of B-tags to I-tags as well as the transition of I-tags to E-tags are however less obvious. For the B-tag to I-tag transition, the entity types "dataset", "method", "resource" and "solution" show high transition scores, albeit being not as high as the I-tag to I-tag transition. Whereas for the I-tag to E-tag transition, only the entity types "research problem", "solution" and "tool" could be seen having high scores for the transition. This could be caused by the length of the entity in the sentence being processed. Entities types such as "research problem" and "solution" tend to have longer spans in the data and hence their transitions of I-tag to I-tag as well as I-tag to E-tag are given high scores. Thus, not all transitions of IOBES-tags for the entity types for the CS-NER dataset can be modeled by the CRF layer. We speculate that the suboptimal hyperparameters chosen for fine-tuning the models led to the incomplete modelling of the IOBES-tag sequences in its predictions, as shown in Tables A.6 and A.4, which show the hyperparameters that gave the best micro F1 results for models with and without the CRF layer, respectively. It requires more training epochs to achieve optimal micro F1 results for the models with the CRF layer. A higher number of training epochs may improve its performance.

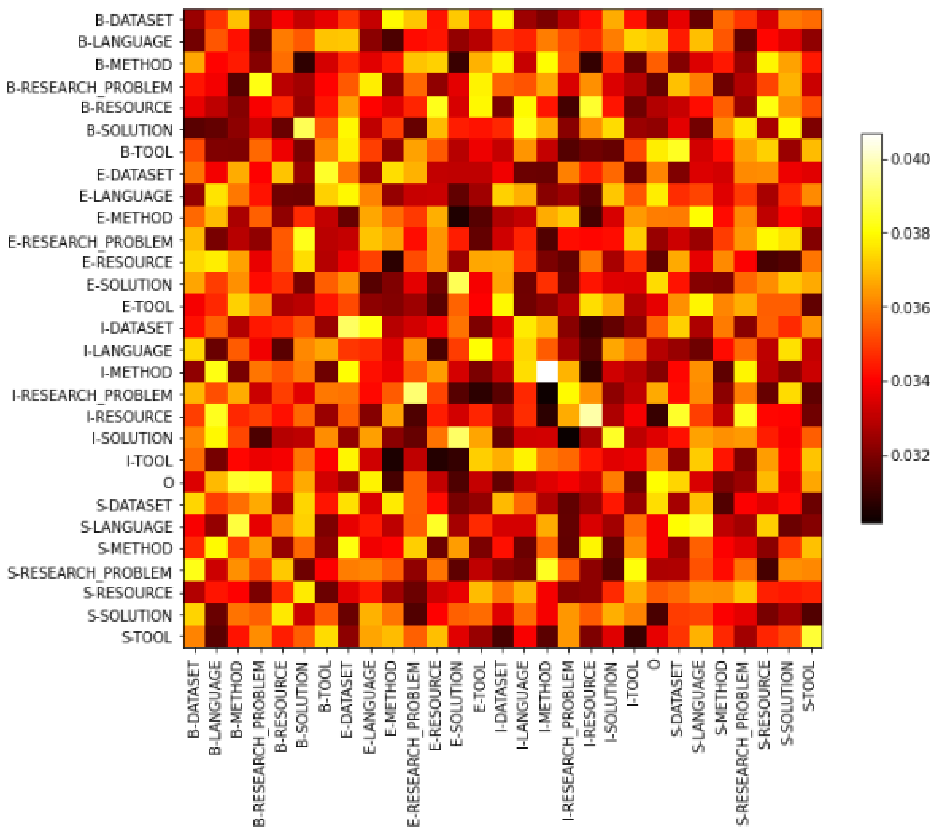


Figure 4.3: Transition scores in CRF layer of PCSciBERT(cased)+CRF for CS-NER dataset.

Due to the larger number of available tags in the WIESP dataset compared to CS-NER dataset, the distribution of the transition scores in the CRF layer is less clear. Nevertheless, it can be seen from Figure 4.4 that high scores for the I-tag to I-tag transitions for the respective entity types are more prominent than I-tag to B-tag transitions, as these are less frequent in the dataset, though a B-tag can appear directly after an I-tag when a seperate entity type is detected directly after in the text. Similar observations can be made when comparing the B-tag to I-tag transitions with the B-tag to B-tag transitions, as it can be illustrated in Figure 4.4 that there are higher scores for the transition of the B-tag to I-tag than there are for the B-tag to B-tag transition. Similarly, mostly 8 training epochs were required to achieve the best micro F1 results for the models with CRF layers when fine-tuning for the WIESP dataset, as show in Table A.7. Possible improvements in micro F1 scores could be achieved if higher number of training epochs used. Fine-tuning of both datasets on higher number of epochs was not resumed due to the longer durations needed to train the models. This should be further studied in future work.

Though it has been reported in previous studies that the addition of a CRF layer does improve the performances of a language model on sequence labeling tasks [1, 16, 57], there have also been reports where the addition of a CRF layer does not improve or even degrades the performance of the model. The authors in [57] used a Portuguese BERT model for an NER task and found that the Portuguese **BERT$_{\text{BASE}}$** model with a CRF layer did not improve the the model without a CRF layer when fine-tuned on the task. Instead, it was found that the Portuguese **BERT$_{\text{LARGE}}$** model showed improvements in its NER performance when CRF was added. This could imply that larger models maybe needed to observe an improvement in the performance.

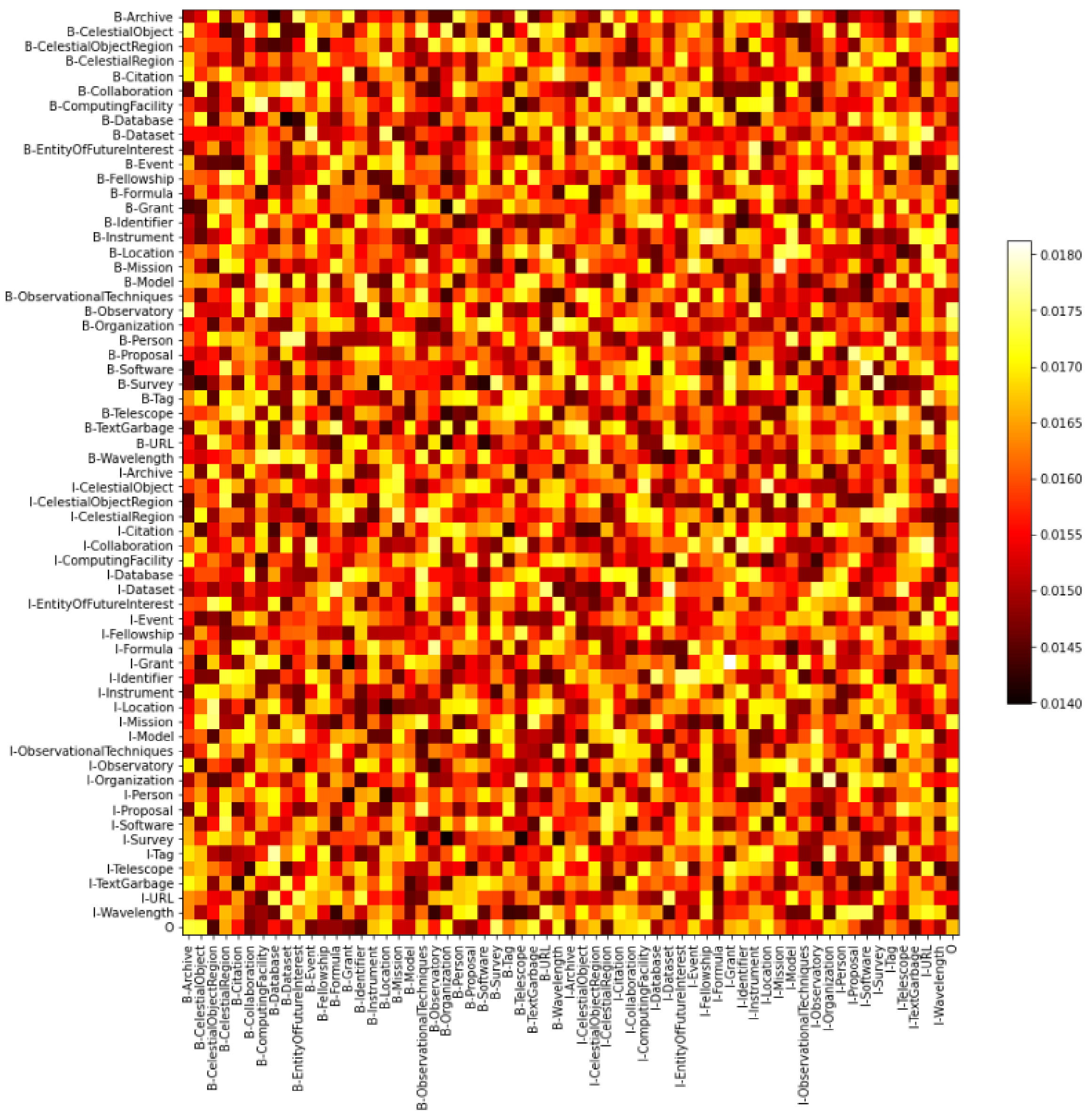
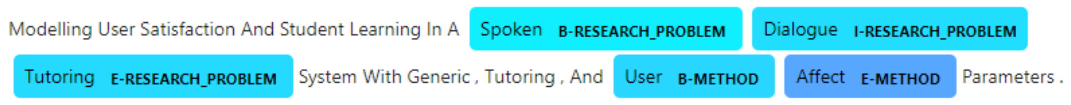


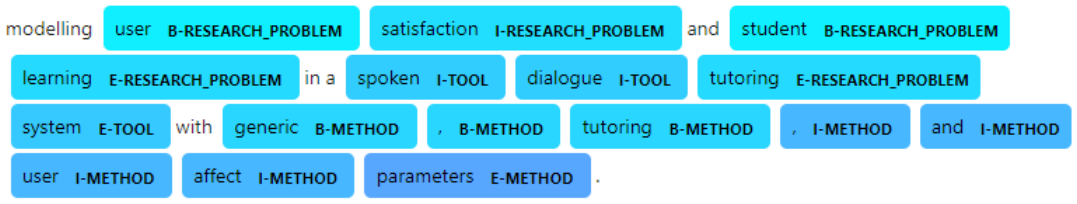
Figure 4.4: Transition scores in CRF layer of PCSciBERT(cased)+CRF for WIESP dataset.

The outputs of the NER models can also be visualized with the help of the Python library displaCy. Figures 4.5 and 4.6 illustrate the example outputs from models for the CS-NER and WIESP dataset, respectively. Before applying the displaCy rendering functions to visualize the entities, words which were broken down into subwords by the models respective tokenizers were merged back together. The corresponding labels were assigned to the merged word, since the subwords have the same label. Hence, the examples shown in Figures 4.5 and 4.6 do not contain any prefix "##", which denotes that a word has been broken down into subwords by the tokenizer. A sample sentence from the test dataset shown in Figure 4.5 illustrates that although PC-

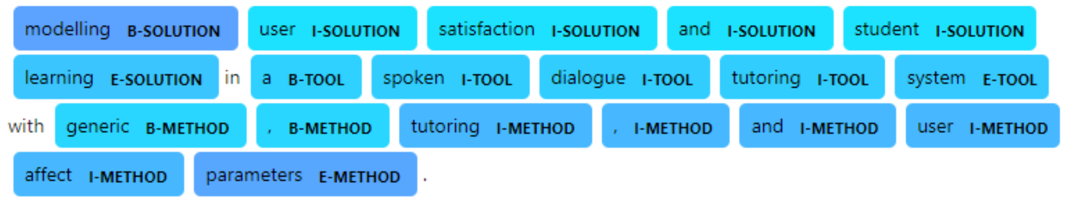
SciBERT(cased)+CRF learned the intrinsic order of the IOBES-scheme for entity types "solution", "tool" and "method", its output had more mismatches to the true labels when compared to the predicted entities from PCSciBERT(cased).


Modelling User Satisfaction And Student Learning In A Spoken B-RESEARCH_PROBLEM Dialogue I-RESEARCH_PROBLEM Tutoring E-RESEARCH_PROBLEM System With Generic , Tutoring , And User B-METHOD Affect E-METHOD Parameters .

(a) Ground truth labels


modelling user B-RESEARCH_PROBLEM satisfaction I-RESEARCH_PROBLEM and student B-RESEARCH_PROBLEM learning E-RESEARCH_PROBLEM in a spoken I-TOOL dialogue I-TOOL tutoring E-RESEARCH_PROBLEM system E-TOOL with generic B-METHOD , B-METHOD tutoring B-METHOD , I-METHOD and I-METHOD user I-METHOD affect I-METHOD parameters E-METHOD .

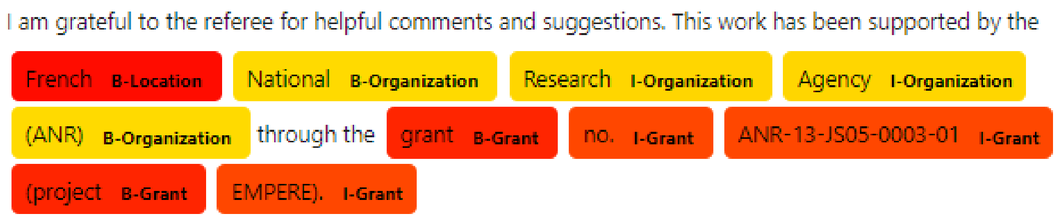
(b) Prediction output from PCSciBERT(cased)


modelling B-SOLUTION user I-SOLUTION satisfaction I-SOLUTION and I-SOLUTION student I-SOLUTION learning E-SOLUTION in a B-TOOL spoken I-TOOL dialogue I-TOOL tutoring I-TOOL system E-TOOL with generic B-METHOD , B-METHOD tutoring I-METHOD , I-METHOD and I-METHOD user I-METHOD affect I-METHOD parameters E-METHOD .

(c) Prediction output from PCSciBERT(cased)+CRF

Figure 4.5: Example predictions from the test dataset of the CS-NER dataset from the best performing models with and without the CRF layer

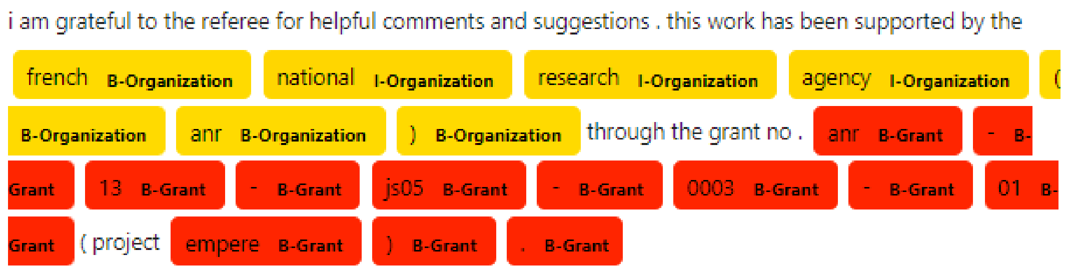
Both PCSciBERT(cased)+CRF and PCSciBERT(cased) achieved similar results in their micro F1 scores with PCSciBERT(cased) having 0.43% more than PCSciBERT(cased)+CRF. Thus, their predictions are also similar. Although both PCSciBERT(cased) and PCSciBERT(cased)+CRF detected similar entities in the example input sentence in Figure 4.6, PCSciBERT(cased)+CRF mispredicted the closing parenthesis ")" in the sentence with the tag "B-Instrument". It can also be seen in the outputs that PCSciBERT(cased) has already learned the intrinsic sequence of the IOB2-tagging scheme. Hence, this could imply that the CRF layer is redundant for the NER task for these datasets.



I am grateful to the referee for helpful comments and suggestions. This work has been supported by the

French B-Location National B-Organization Research I-Organization Agency I-Organization (ANR) B-Organization through the grant B-Grant no. I-Grant ANR-13-JS05-0003-01 I-Grant (project B-Grant EMPERE). I-Grant

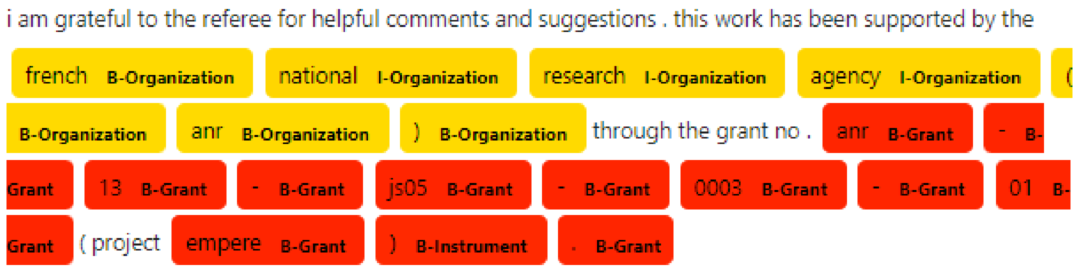
(a) Ground truth labels



i am grateful to the referee for helpful comments and suggestions . this work has been supported by the

french B-Organization national I-Organization research I-Organization agency I-Organization (B-Organization anr B-Organization) B-Organization through the grant no . anr B-Grant - B-Grant 13 B-Grant - B-Grant js05 B-Grant - B-Grant 0003 B-Grant - B-Grant 01 B-Grant (project empere B-Grant) B-Grant . B-Grant

(b) Prediction output from PCSciBERT(cased)



i am grateful to the referee for helpful comments and suggestions . this work has been supported by the

french B-Organization national I-Organization research I-Organization agency I-Organization (B-Organization anr B-Organization) B-Organization through the grant no . anr B-Grant - B-Grant 13 B-Grant - B-Grant js05 B-Grant - B-Grant 0003 B-Grant - B-Grant 01 B-Grant (project empere B-Grant) B-Instrument . B-Grant

(c) Prediction output from PCSciBERT(cased)+CRF

Figure 4.6: Example predictions from the test dataset of the WIESP dataset from the best performing models with and without the CRF layer

5 CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

In the first part of this thesis, we have pre-trained BERT language models initiated with the cased and uncased variants of **BERT$_{\textbf{BASE}}$** and **SciBERT** on scientific texts in the computer science and physics domain collected from the arXiv open repository. We named the models pre-trained with **BERT$_{\textbf{BASE}}$** weights as PCBERT and those pre-trained with **SciBERT** weights as PCSciBERT. To the best of our knowledge this is the first report of domain-specific BERT models pre-trained on computer science and physics texts. The results of pre-training have shown that the pseudo-perplexities of the models were reduced and hence show that the models pre-trained in this work have gained a better understanding of modelling the texts in both of theses domains, although semantically irrelevant placeholder texts were present in the text corpus for pre-training.

In the subsequent part of this thesis, we have evaluated the downstream performance of the pre-trained models on NER tasks with the help of the CS-NER dataset for computer science and WIESP dataset for physics. All models pre-trained on the computer science and physics text corpus in this thesis achieved higher micro F1 scores than their original counterparts, though the improvements of the micro F1 scores for the CS-NER dataset was lower than that for the WIESP dataset. The application of a CRF layer on the pre-trained models were also investigated in this thesis. After finetuning the models with the CRF layer, we found that although the pre-trained models still performed better than their original models, the BERT models with CRF layers performed worse than the BERT models without CRF layers. We assumed this could be due to the lack of training epochs in order to achieve optimial micro F1 scores. Further visual inspection of the transition scores of the CRF layer shows that although some transitions of the tagging schemes were learned by the model, not all of the transitions could be modeled properly by the CRF layer. Nevertheless, our findings show that PCSciBERT(cased) performed the best for all datasets, regardless on whether the CRF layer was applied.

5.2 FUTURE WORK

Future research should consider investigating the performance of PC-SciBERT(cased) on other downstream tasks suchs as text classification and relation extraction, the latter providing more useful information when building semantic knowledge graphs that could ease the search of related scientific articles in the related domains. Datasets, from which the CS-NER dataset was derived from, contained other datasets which offered annotated relations and their corresponding entities. Among them is the SciERC dataset, which contains annotations for identifying and classifying entities, relations and coreference clusters in scientific articles [33]. Furthermore, this thesis shows the lack of downstream NLP tasks in the computer science and physics domains. Hence, this work warrants the need for further development and curation of datasets related to these domains. Another interesting research question for future research that can be derived from the extracted computer science and physics texts in this work is whether the addition of the structural features of mathematical formulas would lead to better information retrieval performances. A recent modified version of the BERT model known as MathBERT can take in mathematical formulas, extracted in operator tree form, as well as its respective context to improve the mathematical information retrieval of texts [42]. This would naturally contribute to the computer science and physics domains, where equations and algorithms, often represented with mathematical notation, are used frequently.

Though most studies have found that the addition of the CRF to the BERT layer does generally improve the performance on downstream sequence labeling tasks [1, 16], further research should examine why in some cases this might not be true, as shown in this work and in [57].

Part II

APPENDIX

A FIGURES AND TABLES

Table A.1: Extracted documents from ArXiv and their corresponding categories.

Categories	**No. of documents (extracted)**	**No. of documents (not extracted)**
Computer Science	449056	4510
Physics: Astrophysics	304273	3634
Physics: Condensed Matter	310916	3977
Physics: General Relativity and Quantum Cosmology	54635	956
Physics:High Energy Physics-Experiment	43492	460
Physics: High Energy Physics-Lattice	19926	346
Physics: High Energy Physics-Phenomology	102473	2154
Physics: High Energy Physics-Theory	58068	1175
Physics: Nonlinear Sciences	9698	119
Physics: Nuclear Experiment	12301	120
Physics:Nuclear Theory	18925	354
Physics: Physics	116553	1348
Physics: Quantum Physics	60345	881
Total	**1560661**	**20034**

Table A.2: Counts of annotated entities in CS-NER dataset. They do not reflect the unique set of annotated entities. Values without parenthesis indicate the count and values in parenthesis indicate the percentage with regard to the total count. The underlined values represent the minimum and the values in bold represent the maximum.

Entity	**Dataset split**		
	train	**development**	**test**
dataset	<u>882 (1.62)</u>	<u>39 (1.2)</u>	<u>228 (1.15)</u>
language	1141 (2.09)	50 (1.54)	499 (2.51)
method	8854 (16.24)	574 (17.63)	2768 (13.94)
research problem	15646 (28.7)	989 (30.37)	4070 (20.5)
resource	7346 (13.48)	439 (13.48)	3226 (16.26)
solution	**18924 (34.72)**	**1072 (32.92)**	**8316 (41.89)**
tool	1718 (3.15)	93 (2.86)	743 (3.74)

Table A.3: Counts of annotated entities in WIESP dataset. They do not reflect the unique set of annotated entities. Values without parenthesis indicate the count and values in parenthesis indicate the percentage with regard to the total count. The underlined values represent the minimum and the values in bold represent the maximum

Entity	**Dataset split**		
	train	**development**	**test**
Archive	192 (0.47)	153 (0.46)	359 (0.58)
CelestialObject	2940 (7.14)	2285 (6.94)	3609 (5.86)
CelestialObjectRegion	265 (0.64)	150 (0.46)	723 (1.17)
CelestialRegion	158 (0.38)	102 (0.31)	209 (0.34)
Citation	6360 (15.45)	4820 (14.64)	8621 (13.99)
Collaboration	306 (0.74)	238 (0.72)	428 (0.69)
ComputingFacility	399 (0.97)	360 (1.09)	607 (0.99)

Table A.3 (continued)

Entity	**Dataset split**		
	train	**development**	**test**
Database	256 (0.62)	199 (0.6)	342 (0.55)
Dataset	328 (0.8)	222 (0.67)	516 (0.84)
EntityOfFutureInterest	61 (0.15)	52 (0.16)	435 (0.71)
Event	45 (0.11)	37 (0.11)	59 (0.1)
Fellowship	411 (1.0)	326 (0.99)	607 (0.99)
Formula	2088 (5.07)	1541 (4.68)	3452 (5.6)
Grant	3478 (8.45)	2834 (8.61)	5259 (8.53)
Identifier	112 (0.27)	94 (0.29)	180 (0.29)
Instrument	714 (1.73)	683 (2.07)	1102 (1.79)
Location	1404 (3.41)	1157 (3.52)	2256 (3.66)
Mission	110 (0.27)	105 (0.32)	204 (0.33)
Model	1800 (4.37)	1412 (4.29)	3208 (5.21)
ObservationalTechniques	124 (0.3)	102 (0.31)	91 (0.15)
Observatory	873 (2.12)	735 (2.23)	1326 (2.15)
Organization	**7448 (18.1)**	**6054 (18.39)**	**11399 (18.5)**
Person	3916 (9.51)	3267 (9.93)	6085 (9.87)
Proposal	100 (0.24)	76 (0.23)	180 (0.29)
Software	1017 (2.47)	839 (2.55)	1386 (2.25)
Survey	885 (2.15)	620 (1.88)	1188 (1.93)
Tag	66 (0.16)	53 (0.16)	115 (0.19)
Telescope	1573 (3.82)	1257 (3.82)	2346 (3.81)
TextGarbage	52 (0.13)	47 (0.14)	283 (0.46)
URL	294 (0.71)	227 (0.69)	405 (0.66)

Table A.3 (continued)

Entity	Dataset split		
	train	development	test
Wavelength	3384 (8.22)	2869 (8.72)	4643 (7.53)

Table A.4: Optimal hyperparameter settings chosen for fine-tuning models without CRF layer for CS-NER dataset.

Models	**No. of Epochs**	**Learning rate**
BERT$_{\textbf{BASE}}$(uncased)	4	1×10^{-4}
BERT$_{\textbf{BASE}}$(cased)	3	3×10^{-4}
SciBERT(uncased)	3	1×10^{-4}
SciBERT(cased)	3	1×10^{-4}
PCBERT(uncased)	2	1×10^{-4}
PCBERT(cased)	4	1×10^{-4}
PCSciBERT(uncased)	3	1×10^{-4}
PCSciBERT(cased)	2	1×10^{-4}

Table A.5: Optimal hyperparameter settings chosen for fine-tuning models without CRF layer for the WIESP dataset.

Models	No. of Epochs	Learning rate
BERT$_{\textbf{BASE}}$(uncased)	8	1×10^{-4}
BERT$_{\textbf{BASE}}$(cased)	8	1×10^{-4}
SciBERT(uncased)	8	1×10^{-4}
SciBERT(cased)	7	1×10^{-4}
PCBERT(uncased)	8	1×10^{-4}
PCBERT(cased)	8	1×10^{-4}
PCSciBERT(uncased)	8	1×10^{-4}
PCSciBERT(cased)	7	1×10^{-4}

Table A.6: Optimal hyperparameter settings chosen for fine-tuning models with CRF layer for CS-NER dataset.

Models	No. of Epochs	Learning rate
BERT$_{\textbf{BASE}}$(uncased)+CRF	8	3×10^{-5}
BERT$_{\textbf{BASE}}$(cased)+CRF	8	3×10^{-5}
SciBERT(uncased)+CRF	8	3×10^{-5}
SciBERT(cased)+CRF	8	3×10^{-5}
PCBERT(uncased)+CRF	8	3×10^{-5}
PCBERT(cased)+CRF	8	3×10^{-5}
PCSciBERT(uncased)+CRF	7	3×10^{-5}
PCSciBERT(cased)+CRF	7	3×10^{-5}

Table A.7: Optimal hyperparameter settings chosen for fine-tuning models with CRF layer for WIESP dataset.

Models	**No. of Epochs**	**Learning rate**
BERT$_{\text{BASE}}$(uncased)+CRF	8	3×10^{-4}
BERT$_{\text{BASE}}$(cased)+CRF	8	1×10^{-4}
SciBERT(uncased)+CRF	6	3×10^{-4}
SciBERT(cased)+CRF	8	3×10^{-4}
PCBERT(uncased)+CRF	8	1×10^{-4}
PCBERT(cased)+CRF	6	1×10^{-4}
PCSciBERT(uncased)+CRF	8	1×10^{-4}
PCSciBERT(cased)+CRF	8	1×10^{-4}

Table A.8: Micro F1 scores of **BERT$_{\text{BASE}}$** and **SciBERT** models with CRF layer fine-tuned on CS-NER dataset.

Models	**Micro F1**
BERT$_{\text{BASE}}$(cased)+CRF	70.29
BERT$_{\text{BASE}}$(uncased)+CRF	70.71
SciBERT(cased)+CRF	69.52
SciBERT(uncased)+CRF	68.96

Table A.9: Micro F1 scores of **BERT$_{\text{BASE}}$** and **SciBERT** models with CRF layer fine-tuned on WIESP dataset.

Models	**Micro F1**
BERT$_{\text{BASE}}$(cased)+CRF	75.25
BERT$_{\text{BASE}}$(uncased)+CRF	77.11
SciBERT(cased)+CRF	78.77
SciBERT(uncased)+CRF	78.59

BIBLIOGRAPHY

[1] Muhammad Saleh Al-Qurishi and Riad Souissi. „Arabic Named Entity Recognition Using Transformer-based-CRF Model.“ In: *Proceedings of the 4th International Conference on Natural Language and Speech Processing (ICNLSP 2021)*. Trento, Italy: Association for Computational Linguistics, 2021, pp. 262–271. URL: https://aclanthology.org/2021.icnlsp-1.31.

[2] Dogu Araci. *FinBERT: Financial Sentiment Analysis with Pre-trained Language Models*. 2019. DOI: 10.48550/ARXIV.1908.10063. URL: https://arxiv.org/abs/1908.10063.

[3] Stephane Aroca-Ouellette and Frank Rudzicz. *On Losses for Modern Language Models*. 2020. arXiv: 2010.01694 [cs.CL].

[4] Iz Beltagy, Kyle Lo, and Arman Cohan. „SciBERT: A Pretrained Language Model for Scientific Text.“ In: (2019). DOI: 10.48550/ARXIV.1903.10676. URL: https://arxiv.org/abs/1903.10676.

[5] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. „A Neural Probabilistic Language Model.“ In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich, and V. Tresp. Vol. 13. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf.

[6] Gianni Brauwers and Flavius Frasincar. „A General Survey on Attention Mechanisms in Deep Learning.“ In: *Transactions on Knowledge and Data Engineering* (2021). DOI: 10.1109/tkde.2021.3126456. URL: https://arxiv.org/abs/2203.14263.

[7] Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. *LEGAL-BERT: The Muppets straight out of Law School*. 2020. DOI: 10.48550/ARXIV.2010.02559. URL: https://arxiv.org/abs/2010.02559.

[8] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL].

[9] *DEAL: Detecting Entities in the Astrophysics Literature*. https://ui.adsabs.harvard.edu/WIESP/2022/SharedTasks. Accessed: 2023-14-03.

[10] Jennifer D'Souza and Soeren Auer. *Pattern-based Acquisition of Scientific Entities from Scholarly Article Titles*. 2021. DOI: 10.48550/ARXIV.2109.00199. URL: https://arxiv.org/abs/2109.00199.

[11] Jennifer D'Souza, Sören Auer, and Ted Pedersen. „SemEval-2021 Task 11: NLPContributionGraph - Structuring Scholarly NLP Contributions for a Research Knowledge Graph." In: *Proceedings of the 15th International Workshop on Semantic Evaluation (SemEval-2021)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 364–376. DOI: 10.18653/v1/2021.semeval-1.44. URL: https://aclanthology.org/2021.semeval-1.44.

[12] Jennifer D'Souza and Sören Auer. *Computer Science Named Entity Recognition in the Open Research Knowledge Graph*. 2022. DOI: 10.48550/ARXIV.2203.14579. URL: https://arxiv.org/abs/2203.14579.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805.

[14] Andrea Galassi, Marco Lippi, and Paolo Torroni. „Attention in Natural Language Processing." In: *IEEE Transactions on Neural Networks and Learning Systems* 32.10 (2021), pp. 4291–4308. DOI: 10.1109/tnnls.2020.3019893. URL: https://arxiv.org/abs/1902.02181.

[15] Sonal Gupta and Christopher Manning. „Analyzing the Dynamics of Research by Extracting Key Aspects of Scientific Papers." In: *Proceedings of 5th International Joint Conference on Natural Language Processing*. Chiang Mai, Thailand: Asian Federation of Natural Language Processing, Nov. 2011, pp. 1–9. URL: https://aclanthology.org/I11-1001.

[16] Tanishq Gupta, Mohd Zaki, N. M. Anoop Krishnan, and Mausam. *MatSciBERT: A Materials Domain Language Model for Text Mining and Information Extraction*. 2021. DOI: 10.48550/ARXIV.2109.15290. URL: https://arxiv.org/abs/2109.15290.

[17] Sepp Hochreiter and Jürgen Schmidhuber. „Long Short-term Memory." In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[18] Kexin Huang, Jaan Altosaar, and Rajesh Ranganath. *ClinicalBERT: Modeling Clinical Notes and Predicting Hospital Readmission*. 2019. DOI: 10.48550/ARXIV.1904.05342. URL: https://arxiv.org/abs/1904.05342.

[19] Shu Huang and Jacqueline M Cole. „BatteryBERT: A Pretrained Language Model for Battery Database Enhancement.“ In: *J. Chem. Inf. Model.* (2022), DOI: 10.1021/acs.jcim.2c00035. DOI: 10.1021/acs.jcim.2c00035. URL: DOI:10.1021/acs.jcim.2c00035.

[20] Shu Huang and Jacquelinse Cole. „A database of battery materials auto-generated using ChemDataExtractor.“ In: (July 2020). DOI: 10.6084/m9.figshare.11888115.v2. URL: https://figshare.com/articles/dataset/A_database_of_battery_materials_auto-generated_using_ChemDataExtractor/11888115.

[21] *Hugging Face*. https://github.com/huggingface. Accessed: 2023-12-03.

[22] Kun Jing and Jungang Xu. *A Survey on Neural Network Language Models*. 2019. DOI: 10.48550/ARXIV.1906.03591. URL: https://arxiv.org/abs/1906.03591.

[23] Julijan Jug, Ajda Lampe, Vitomir Štruc, and Peter Peer. „Body Segmentation Using Multi-task Learning.“ In: *2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. 2022, pp. 060–068. DOI: 10.1109/ICAIIC54071.2022.9722662.

[24] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: N-gram Language Models*. 2023.

[25] Amirhossein Kazemnejad. In: (2023). URL: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/.

[26] John D. Lafferty, Andrew McCallum, and Fernando Pereira. „Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data.“ In: *International Conference on Machine Learning*. 2001.

[27] Jieh-Sheng Lee and Jieh Hsiang. *PatentBERT: Patent Classification with Fine-Tuning a pre-trained BERT Model*. 2019. DOI: 10.48550/ARXIV.1906.02124. URL: https://arxiv.org/abs/1906.02124.

[28] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. „BioBERT: a pre-trained biomedical language representation model for biomedical text mining.“ In: *Bioinformatics* 36.4 (2019). Ed. by Jonathan Wren, pp. 1234–1240. DOI: 10.1093/bioinformatics/btz682. URL: https://doi.org/10.1093\%2Fbioinformatics\%2Fbtz682.

[29] Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. *A Survey on Deep Learning for Named Entity Recognition*. 2020. arXiv: 1812.09449 [cs.CL].

[30] Qi Liu, Matt J. Kusner, and Phil Blunsom. *A Survey on Contextual Embeddings*. 2020. DOI: 10.48550/ARXIV.2003.07278. URL: https://arxiv.org/abs/2003.07278.

[31] Wenbin Liu, Bojian Wen, Shang Gao, Jiesheng Zheng, and Yinlong Zheng. „A multi-label text classification model based on ELMo and attention." In: *MATEC Web of Conferences* 309 (Jan. 2020), p. 03015. DOI: 10.1051/matecconf/202030903015.

[32] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. DOI: 10.48550/ARXIV.1907.11692. URL: https://arxiv.org/abs/1907.11692.

[33] Yi Luan, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. „Multi-Task Identification of Entities, Relations, and Coreference for Scientific Knowledge Graph Construction." In: (2018). DOI: 10.48550/ARXIV.1808.09602. URL: https://arxiv.org/abs/1808.09602.

[34] Renqian Luo, Liai Sun, Yingce Xia, Tao Qin, Sheng Zhang, Hoifung Poon, and Tie-Yan Liu. „BioGPT: generative pre-trained transformer for biomedical text generation and mining." In: *Briefings in Bioinformatics* 23.6 (2022). DOI: 10.1093/bib/bbac409. URL: https://doi.org/10.1093%2Fbib%2Fbbac409.

[35] Kanchan M.Tarwani and Swathi Edem. „Survey on Recurrent Neural Network in Natural Language Processing." In: *International Journal of Engineering Trends and Technology* 48 (June 2017), pp. 301–304. DOI: 10.14445/22315381/IJETT-V48P253.

[36] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. *Learned in Translation: Contextualized Word Vectors*. 2017. DOI: 10.48550/ARXIV.1708.00107. URL: https://arxiv.org/abs/1708.00107.

[37] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. DOI: 10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781.

[38] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. „Recurrent neural network based language model." In: vol. 2. Jan. 2010, pp. 1045–1048.

[39] Hiroki Nakayama. *seqeval: A Python framework for sequence labeling evaluation*. Software available from https://github.com/chakki-works/seqeval. 2018. URL: https://github.com/chakki-works/seqeval.

[40] *Natural Language Toolkit*. https://www.nltk.org/. Accessed: 2023-03-07.

[41] *PapersWithCode*. https://paperswithcode.com/. Accessed: 2023-03-14.

[42] Shuai Peng, Ke Yuan, Liangcai Gao, and Zhi Tang. *MathBERT: A Pre-Trained Model for Mathematical Formula Understanding*. 2021. arXiv: 2105.00377 [cs.CL].

[43] Jeffrey Pennington, Richard Socher, and Christopher Manning. „GloVe: Global Vectors for Word Representation.“ In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

[44] Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. *Semi-supervised sequence tagging with bidirectional language models*. 2017. DOI: 10.48550/ARXIV.1705.00108. URL: https://arxiv.org/abs/1705.00108.

[45] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke 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, June 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: https://aclanthology.org/N18-1202.

[46] *PyMuPDF*. https://pymupdf.readthedocs.io/en/latest/. Accessed: 2023-03-07.

[47] *Python Transformers library*. https://pypi.org/project/transformers/. Accessed: 2023-02-08.

[48] Nils Reimers and Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. 2019. DOI: 10.48550/ARXIV.1908.10084. URL: https://arxiv.org/abs/1908.10084.

[49] Michele Resta, Daniele Arioli, Alessandro Fagnani, and Giuseppe Attardi. „Transformer Models for Question Answering at BioASQ 2019.“ In: Mar. 2020, pp. 711–726. ISBN: 978-3-030-43886-9. DOI: 10.1007/978-3-030-43887-6_63.

[50] Julian Salazar, Davis Liang, Toan Q. Nguyen, and Katrin Kirchhoff. „Masked Language Model Scoring." In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020. DOI: 10.18653/v1/2020.acl-main.240. URL: https://aclanthology.org/2020.acl-main.240/.

[51] Sivasurya Santhanam. *Context based Text-generation using LSTM networks*. 2020. arXiv: 2005.00048 [cs.CL].

[52] Nikunj Saunshi, Sadhika Malladi, and Sanjeev Arora. *A Mathematical Exploration of Why Language Models Help Solve Downstream Tasks*. 2020. DOI: 10.48550/ARXIV.2010.03648. URL: https://arxiv.org/abs/2010.03648.

[53] Mike Schuster and Kaisuke Nakajima. „Japanese and Korean voice search." In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012, pp. 5149–5152. DOI: 10.1109/ICASSP.2012.6289079.

[54] *SentencePiece library*. https://github.com/google/sentencepiece. Accessed: 2023-03-08.

[55] Xinying Song, Alex Salcianu, Yang Song, Dave Dopson, and Denny Zhou. *Fast WordPiece Tokenization*. 2020. DOI: 10.48550/ARXIV.2012.15524. URL: https://arxiv.org/abs/2012.15524.

[56] Daniel Soutner and Luděk Müller. „Application of LSTM Neural Networks in Language Modelling." In: *Text, Speech, and Dialogue*. Ed. by Ivan Habernal and Václav Matoušek. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 105–112. ISBN: 978-3-642-40585-3.

[57] Fábio Souza, Rodrigo Nogueira, and Roberto Lotufo. *Portuguese Named Entity Recognition using BERT-CRF*. 2020. arXiv: 1909.10649 [cs.CL].

[58] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. „LSTM Neural Networks for Language Modeling." In: Sept. 2012. DOI: 10.21437/Interspeech.2012-65.

[59] Charles Sutton and Andrew McCallum. *An Introduction to Conditional Random Fields*. 2010. arXiv: 1011.4088 [stat.ML].

[60] Sho Takase, Shun Kiyono, Sosuke Kobayashi, and Jun Suzuki. *On Layer Normalizations and Residual Connections in Transformers*. 2022. DOI: 10.48550/ARXIV.2206.00330. URL: https://arxiv.org/abs/2206.00330.

[61] Amalie Trewartha, Nicholas Walker, Haoyan Huo, Sanghoon Lee, Kevin Cruse, John Dagdelen, Alexander Dunn, Kristin A. Persson, Gerbrand Ceder, and Anubhav Jain. „Quantifying the advantage of domain-specific pre-training on named entity recognition tasks in materials science.“ In: *Patterns* 3.4 (2022), p. 100488. ISSN: 2666-3899. DOI: https://doi.org/10.1016/j.patter.2022.100488. URL: https://www.sciencedirect.com/science/article/pii/S2666389922000733.

[62] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[63] Li Wang, Rongzhi Gu, Nuo Chen, and Yuexian Zou. *Text Anchor Based Metric Learning for Small-footprint Keyword Spotting*. 2021. arXiv: 2108.05516 [cs.SD].

[64] Shuohang Wang and Jing Jiang. *Learning Natural Language Inference with LSTM*. 2015. DOI: 10.48550/ARXIV.1512.08849. URL: https://arxiv.org/abs/1512.08849.

[65] *Workshop on Information Extraction from Scientific Publications NER Dataset*. https://huggingface.co/datasets/adsabs/WIESP2022-NER. Accessed: 2023-14-03.

[66] *Workshop on Information Extraction from Scientific Publications*. https://ui.adsabs.harvard.edu/WIESP/. Accessed: 2023-14-03.

[67] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. *Comparative Study of CNN and RNN for Natural Language Processing*. 2017. DOI: 10.48550/ARXIV.1702.01923. URL: https://arxiv.org/abs/1702.01923.

[68] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. *Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books*. 2015. DOI: 10.48550/ARXIV.1506.06724. URL: https://arxiv.org/abs/1506.06724.

[69] *arXiv Bulk Data Access*. https://info.arxiv.org/help/bulk_data_s3.html. Accessed: 2023-12-03.

[70] *arXiv Computer Science Archive*. https://arxiv.org/archive/cs/22. Accessed: 2023-02-08.

[71] *arXiv Physics Archive*. https://arxiv.org/archive/physics/22. Accessed: 2023-02-08.

[72] *langdetect Java library (Google)*. https://code.google.com/archive/p/language-detection/. Accessed: 2023-02-07.

[73] *langdetect Python library*. https://pypi.org/project/langdetect/. Accessed: 2023-02-07.