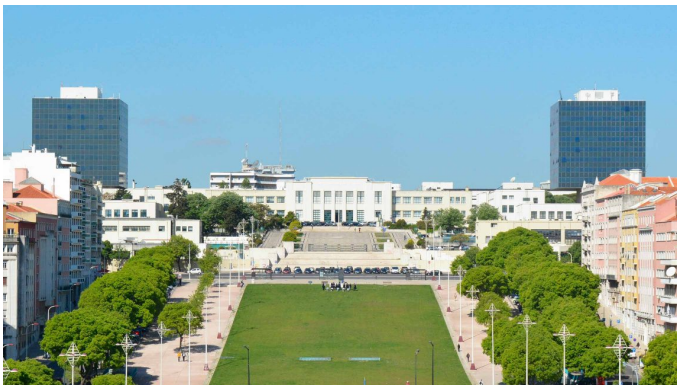
TÉCNICO LISBOA



Natural Language Processing Leveraging Entity-Aware Representations

José Luís Mourão Malaquias

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Prof. Bruno Emanuel da Graça Martins
Dr. Pedro José dos Reis Mota

Examination Committee

Chairperson: Prof. Pedro Filipe Zeferino Aidos Tomás
Supervisor: Prof. Bruno Emanuel da Graça Martins
Member of the Committee: Dr. Fábio Natanael Kepler
Prof. Alberto Abad Gareta

31st March 2023

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

A realização desta dissertação de mestrado contou com importantes apoios e incentivos sem os quais não se teria tornado uma realidade e aos quais estarei eternamente grato. Agradeço também todo o apoio da Unbabel.

Aos meus orientadores, pelo seu acompanhamento, total apoio, disponibilidade, pelo saber que transmitiram, pelas opiniões e críticas, total colaboração no solucionar de dúvidas e problemas que foram surgindo ao longo da realização deste trabalho e por todas as palavras de incentivo.

Ao corpo docente do Instituto Superior Técnico pelo conhecimento transmitido, pela sua colaboração e disponibilidade em tirar duvidas.

Aos amigos que fiz no curso, Henrique, Cantante, Leonor, Martinho, Maggs, Martinha, Alexandra, Manel, Miguel, Megui, entre outros que não menciono o nome mas sabem quem são, que me acompanharam pelos laboratórios, projetos e noitadas de estudo. Amigos que estiveram ao meu lado durante todo o curso, e que me deram força e apoio nos momentos mais difíceis para nunca desistir.

A todos os apontamentos, relatórios de laboratórios e testes resolvidos das Drives de Eletro cedidos por ex alunos, sem os quais não teria obtido certas notas em momentos críticos.

Aos meus padrinhos de curso Inês e Godinho assim como os meus amigos de Lisboa, Diogo, Godinho, Mariana, Jéssica, Pedro, João, Alexandre, Carolina, Lucia, pelo companheirismo, pelos momentos e gargalhadas proporcionadas, lendárias noites na Cervetoria, idas a Belém, e jantaradas por Lisboa, especialmente aquelas em que se enganaram na conta e pagámos menos. Agradeço-lhes terem estado ao meu lado nos bons e nos maus momentos e me terem aconselhado a seguir o caminho me levou até aqui.

Aos meus grandes amigos Figueirenses, Roque, Samuel, Pedro, Miguel, Nuno, entre outros, que acompanharam todo o meu percurso académico e que hoje posso com confiança dizer que são amigos verdadeiros. Foram momentos como as saídas para café, as idas às setas, as sessões de Pro Clubs e Sniper Elite, as idas ao Estádio da Luz, e os jogos lendários no sintético em que o Antero nos carregava; que me distraíram dos momentos mais difíceis que vivi nos últimos anos e me deram as forças necessárias para ultrapassar as várias etapas da minha carreira académica.

Finalmente, tenho consciência que sozinho nada disto teria sido possível, dirijo

um agradecimento muito especial a toda a minha família, sobretudo a minha mãe. Agradeço-lhes por serem modelos de coragem, pelo seu apoio incondicional, incentivo, amizade e paciência demonstrados e total ajuda na superação dos obstáculos que ao longo desta difícil caminhada foram surgindo. A eles e a todas as pessoas referidas aqui dedico este trabalho!

Abstract

Recent Pre-trained Language Models (PLM), based on self-attention and the Transformer neural architecture, produce a contextualized representation of words. The use of these representations has enabled state-of-the-art results in different downstream Natural Language Processing (NLP) tasks. A new line of research focuses on studying how to incorporate named entity components within a PLM. The motivation is that named entities convey essential information that can further improve the performance of a model in a NLP task. However, current entity-focused models face limitations, such as the inability to process long inputs. This is especially noticeable in recent complex tasks which demand good understanding over long sequences of text. In this article, we propose Long-LUKE, a model based on the entity-aware LUKE PLM that is capable of processing long sequences of text. In particular, Long-LUKE obtains interesting results on four well-known datasets: Open Entity, FIGER and DocRED (entity typing), and TACRED and ReDocRED (relation classification).

Keywords: Named Entities, Long Documents, Transformers

Resumo

Recentes Modelos de Linguagem Pré-treinados (MLPs), baseados na atenção própria e na arquitetura Transformer, produzem representações de palavras contextualizadas. A utilização destas representações permitiu atingir resultados estado-de-arte em diferentes tarefas de Processamento de Linguagem Natural (PLN). Uma nova linha de investigação foca-se no estudo de como incoporar nomes de entidades em MLPs. A ideia é que as entidades possuem informação essencial que pode aumentar o desempenho de um modelo numa tarefa de PLN, quando estas são tidas em conta. No entanto, os atuais modelos focados em entidades apresentam limitações, tal como a incapacidade de processar documentos longos. Isto é especialmente notável em tarefas complexas recentes que dependam de uma boa compreensão do contexto ao longo de sequências extensas de texto. Esta dissertação de mestrado apresenta o Long-LUKE, um novo modelo descendente de modelos existentes que lidam com entidades e possuem atenção própria, como é o caso do LUKE. Estes modelos propõem técnicas para solucionar as limitações existentes nas tarefas relacionadas com entidades. Em particular, o nosso modelo obtém resultados interessantes em 4 conjuntos de dados relevantes: OpenEntity, FIGER e DocRED (classificação de entidades), e TACRED e ReDocRED (extração de relações entre entidades).

Keywords: Entidades Mencionadas, Documentos Longos, Transformers, Atenção própria

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Recently, Pre-trained Language Models (PLM) such as Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019a), based on self-attention and the Transformer neural architecture, outperformed previous approaches to modelling sequences of words within Natural Language Processing (NLP) applications, such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks. The self-attention mechanism proposed in the Transformer architecture (Vaswani et al., 2017) was a disruptive idea, as it allowed the modelling of dependencies without regard to their distance in the input or output sequences. Another disruptive idea was to treat words and entities in a given text as different types of tokens (Yamada et al., 2020; Baldini Soares et al., 2019). Properly representing entities (Qin et al., 2021a; Joko et al., 2021) is especially relevant, as many tasks (i.e., Question Answering (QA), Machine Translation, Chatbots) involve identifying these critical elements in the text, like names of people, places, or brands. Furthermore, the model performance may benefit from the awareness of the representation of entities, as it is mainly from those entities that we can locate the subject and context of a document Blei et al. (2003).

Specifically, we show an example in Figure 1.1 to understand why it is relevant to be aware of entities (highlighted words) and their relations within a text. In the example, to understand "Where is the Republic of Ireland," we have to consider the clues jointly: (1) The "United Kingdom" is located in "Europe" in Sentence 1; (2) The "United Kingdom" includes part of the "island of Ireland" in Sentence 2; (3) "Northern Ireland" shares a land border with "Republic of Ireland" in Sentence 3; (4) "Northern Ireland" is one of the countries of "United Kingdom" in Sentence 6. From the example we learn that to

understand an entity we must consider its relations to other entities comprehensively. In the example, the entity "United Kingdom" appearing in Sentences 1, 2, 4 and 6 helps us find the answer. To understand "United Kingdom", we should consider all its connected entities and diverse relations among them, and to understand the relations between pairs of entities, we need to comprehend complex reasoning patterns in the text (e.g., we can infer the "Republic of Ireland" is in "Europe" as it shares borders with a country of "United Kingdom", which is located in "Europe"). Through the relations between entities, we can capture an entity's true meaning. For example, in sentence 5, with a lack of context, the entity "English channel" could have multiple meanings.

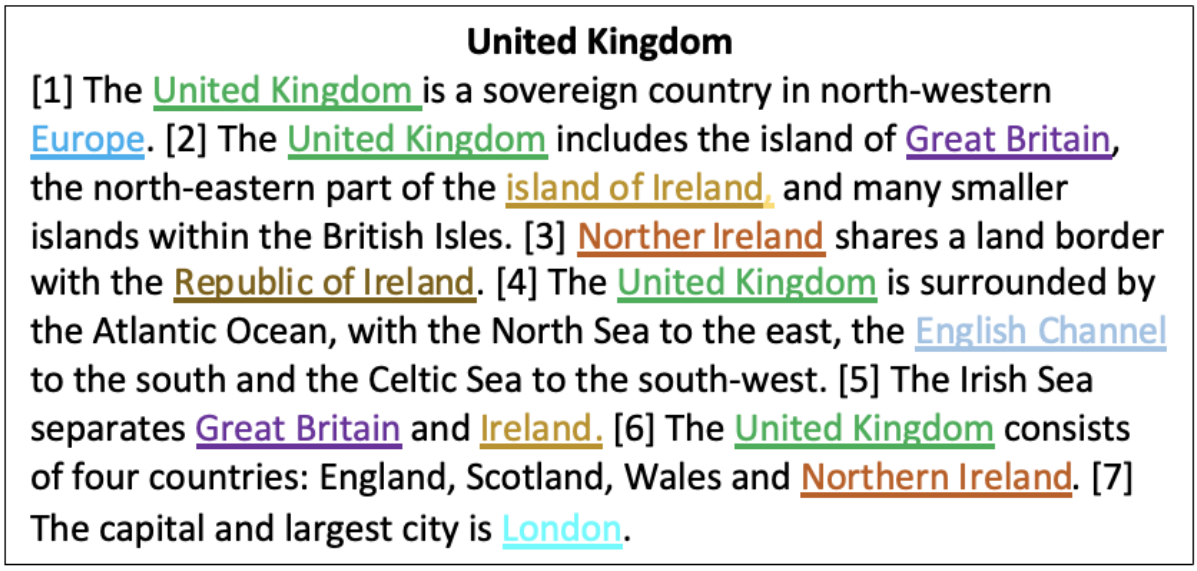


Figure 1.1: An example for a document "United Kingdom", in which all entities are underlined. We highlight the important entities and relations to find out "Where is the Republic of Ireland".

1.2 Research Hypothesis

Our central research hypothesis is to verify whether entity related NLP tasks have improved results when models take into account existing entities as an input, verify whether changing the self-attention pattern benefits the model by increasing the scores on the aforementioned tasks, and also check if it is possible to enable entity-related models to deal with longer input sequences, and decrease the quadratic complexity caused by the model's self-attention.

A model with a good comprehension of the text can perform better on tasks that rely on context, especially if we need to compare multiple relations along a multiple-sentenced document to perceive such knowledge. However, most PLMs do not con-

sider entities and their relations among multiple entities at the document level, whose context involves complex reasoning patterns.

A recently proposed entity-aware model named Language Understanding with Knowledge-based Embeddings (LUKE) (Yamada et al., 2020), which focuses on explicitly modelling entities, significantly improved the performance on entity-related tasks, such as entity typing, relation classification, named entity recognition, and others. This work focused on studying which implementation aspects can benefit from recent studies, making LUKE able to handle larger textual inputs (Beltagy et al., 2020; Zaheer et al., 2020).

1.3 Contributions

This work presents Long-LUKE, i.e., an extended model from LUKE capable of handling longer text sequences. Long-LUKE has a different self-attention pattern from LUKE. Although the entity-focused attention operations remain the same, the textual attention pattern follows the Longformer (Beltagy et al., 2020) mechanism that scales linearly with sequence length. Following prior work on entity-oriented transformers, we evaluate Long-LUKE on various downstream tasks. Our model achieves the same performance for short-sequenced datasets as all baseline models. Additionally, Long-LUKE accomplishes up-to-standard results on both DocRED (Yao et al., 2019) and ReDocRED (Tan et al., 2022) on the entity typing and relation classification tasks, featuring long-sequenced documents.

1.4 Organization of the Dissertation

The remainder of this dissertation is organized in the following manner: Chapter 2 describes related work, and Chapter 3 details the proposed approaches. In contrast, Chapter 4 presents the experimental evaluation methodology and the obtained results. Finally, Chapter 5 summarizes our contributions and discusses future improvements to our approach. The code to reproduce our experiments can be found in this GitHub repository: `https://github.com/joseMalaquias/LongLuke-Jose-Malaquias.git`

Chapter 2

Background

2.1 Fundamental Concepts

This Section describes critical concepts to understand the architectures presented in this report. It starts by detailing how a basic Neural Network (NN) works, how it can be used in machine learning tasks, and what techniques can be used to train and optimize NNs. Afterwards, we will explain some models which are related to this subject, namely the Transformer architecture (Vaswani et al., 2017), the BERT model (Devlin et al., 2019a), and LUKE (Yamada et al., 2020).

2.1.1 Learning with Neural Networks

Neural Networks (NNs) consist of artificial neurons (i.e., perceptrons) assembled into a network capable of performing supervised learning. In theory, they are very good function approximators. Most NNs are organized into layers of nodes and are *feed-forward*, meaning the data moves through them in one direction only, and is organized into layers of nodes. Nodes in the same layer do not connect: they receive data input from nodes in the previous layer and send their output to the next layer. Each connection has an assigned weight multiplied by the input data. The products from the proceeding nodes are added to each other, resulting in a single number passed to an activation function. When training NNs, the parameters (i.e., the weights) are adjusted until the output produced for data has consistent results with pre-existing annotations.

2.1.1.1 General Architecture

The simplest unit of a NN is the perceptron, corresponding to 2 single neurons and where the input x is fed directly to the output linearly via a series of weights, which can be formally defined as:

$$y = x \cdot w + b, \tag{2.1}$$

where y is the *pre-activated* output of the neuron, w represents the connection's weights, and b is a bias term. The bias allows the model to fit the prediction with the data better. The role of each weight is to define how much the dimension x_i stimulates the neuron responsible for the output y.

A single neuron can only learn strictly linear functions. Therefore, if we wish to tackle more complex problems, a possible solution is to stack several of these layers and feed the output of one layer as the input of the next layer. When doing so, we create a Multi-Layer Perceptron (MLP). The layers placed between input and output layers are called *hidden layers*, and they increase the expressive power of the network. As an example, if we consider two hidden layers in our model, $f(x)$ the output of the MLP, and $y^1 = x^1 \cdot w^1 + b^1$ the output of the first layer, we obtain:

$$\begin{aligned} f(x) &= y^2 \\ &= x^2 \cdot w^2 + b^2 \\ &= y^1 \cdot w^2 + b^2 \\ &= \left(x^1 \cdot w^1 + b^1\right) \cdot w^2 + b^2 \\ &= x\left(w^1 w^2\right) + \left(b^1 w^2 + b^2\right) \\ &= x \cdot w + b, \end{aligned} \tag{2.2}$$

where w^n and b^n are, respectively, the weight matrix and bias vector of the layer l^n. Perceptrons can also be combined with an activation function, i.e. a function whose purpose is to introduce non-linearity into the output of a neuron. This helps the NN learn complex data patterns. Within a MLP, the output of a layer can be passed through an activation function that attenuates values below a certain threshold and increases values above it. Standard activation functions are the logistic sigmoid, the REctified Linear Unit (RELU), and the hyperbolic tangent (tanh).

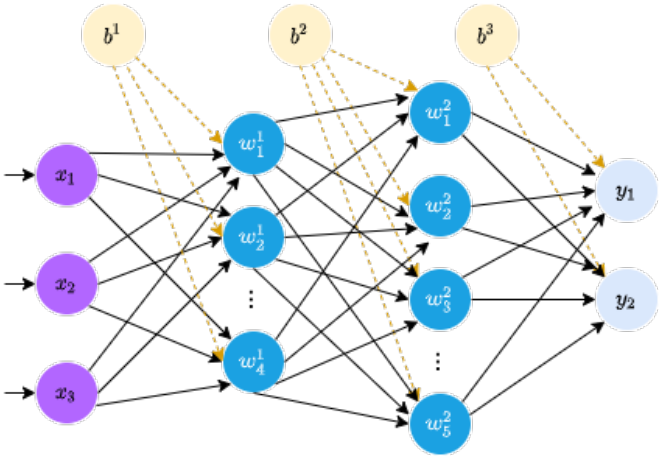


Figure 2.1: Graphical representation of a Multi-Layer Perceptron (feed-forward), with an input layer of dimensionality 3, 2 hidden layers of dimensionality 4 and 5, respectively, and an output layer of dimensionality 2.

After applying an activation function, Equation 2.1 becomes:

$$y = g\left(x \times w + b\right), \tag{2.3}$$

where $g(z)$ is the desired activation function. Figure 2.1 represents a NN and its processes. In the figure, along with an input and an output layer, we have two hidden layers of perceptrons placed between them, which goal is to perform nonlinear transformations of the inputs entered into the network.

Training a NN involves optimizing its parameters (i.e., the connection weight θ) for a target task. In an optimization algorithm, we use *objective function* $J(\theta)$ to evaluate a candidate solution θ. When we search for a possible solution, we wish to minimize the error between expected and produced values. As such, the *objective function* is also referred to as *loss function*. The idea is to quantify the *loss* in the form of a single real number during the training phase to measure how well the NN predicts the output on the training set. A *loss function* is computed for a single training example/input, whereas a *cost function* is the average *loss* over the entire training dataset. A commonly used *cost function* for training neural networks in regression problems is the *Mean Squared Error* (MSE), which is formally expressed as:

$$J(\theta) = \frac{1}{2}\|f(\theta) - y\|^2, \tag{2.4}$$

where $f(\theta)$ is the value returned by the model. In a Neural Network, the loss function is only attached to the output layer, where the predictions come out. In order to propagate

the error to the weights and biases before the output, we use the *back-propagation algorithm* (Rumelhart et al., 1986).

Having established the concepts behind the architecture of a NN and acknowledging that calculating and minimizing a *loss function* for all values over the entire dataset can be challenging, we need a learning algorithm capable of finding the necessary parameter values (biases and weights). The mechanism responsible for updating the parameters is known as the Gradient Descent (GD).

2.1.1.2 Gradient Descent

GD is a popular optimization algorithm used to train machine learning models and NNs. Mathematically, the gradient is a partial derivative concerning its inputs, which measures the change in weights concerning the change in the error. The gradient is seen as the slope of a function. A higher slope means a high gradient, causing the model to learn faster. However, if the slope is zero, the model does not learn.

GD can combine this strategy with multiple algorithms to find a local minimum of a differentiable function. In deep learning, GD is used to find the coefficients that minimize a function $J(\theta)$. Once the initial parameter values ($\theta \in \mathbb{R}^d$) are defined, GD uses calculus to iteratively update the values in the opposite direction of the gradient of the objective function $\Delta_\theta J(\theta)$, to minimize the *cost function*.

Following Equation 2.4 for the parameters corresponding to weights w and biases b, and assuming a regressive problem, we obtain a cost function:

$$\begin{aligned} J(\theta) = J(w,b) &= \frac{1}{2}\|f(w,b) - y\|^2 \\ &= \frac{1}{N}\sum_{i=1}^{n}\left(y_i - (wx_i + b)\right)^2, \end{aligned} \tag{2.5}$$

From this, the gradient is given by:

$$\Delta J(\theta) = \Delta J(w,b) = \begin{bmatrix} \frac{df}{dw} \\ \\ \frac{df}{db} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum -2x_i\left(y_i - (wx_i + b)\right) \\ \\ \frac{1}{N}\sum -2\left(y_i - (wx_i + b)\right) \end{bmatrix}. \tag{2.6}$$

We iterate using the updated parameter values and compute the partial derivatives to solve this gradient through our data points. The values of the new gradient indicate the slope of our cost function at our current parameter values and the direction we should

move to update the parameters. The learning rate η gives the size of the steps we take to reach the local minimum. Combining GD with the back-propagation algorithm, which is based on the chain rule, the calculation of the gradient flows backwards through the network, with the gradient of the final layer of weights being calculated first and the gradient of the first layer of weights being calculated last. Some computations regarding the gradient from one layer are reused in the previous layer. This backwards flow of the error information enables a more efficient calculation of the gradient at each layer in opposition to calculating the gradient of each layer separately.

The back-propagation algorithm starts with a regular forward pass, where the NN takes the input and sends it through all the layers until the output is produced. The prediction error is obtained by comparing the output fed to the loss function with the expected value. Starting from the final layer, the error will be back-propagated inside the NN, updating the parameters of the nodes of each layer along the way.

The three main variants of GD differ in their accuracy, the time they take to update, and the amount of data they require to compute the gradient of the loss function. Batch Gradient Descent is the simplest form of GD where all parameters are updated using the average of the gradients over all the training data. This approach is more straightforward and majorly allows convergence to the global minimum. However, using all the training data in each epoch can be a drawback in cases with many data. Each epoch has only one step of GD. This gradient is given by:

$$\Theta_{t+1} = \Theta_t - \eta \Delta_\Theta J(\Theta_t), \tag{2.7}$$

where Θ are the parameters of the model, weights and biases, $J(.)$ is the loss function, and η is the learning rate, which defines the size of the learning step.

Stochastic Gradient Descent is a variant where the gradient computation is done for each training example separately. It is commonly used because it is computationally the cheapest alternative. In each step, to update the parameters, we only use one instance of the dataset to estimate the gradient. Even though it consumes fewer resources, each update becomes more irregular, as the training example can vary. In this method, we need to tune the learning rate to achieve convergence carefully.

Formally, the method mentioned above is expressed in the following equation:

$$\Theta_{t+1} = \Theta_t - \eta \Delta_\Theta J(\Theta_t, x_i, y_i), \tag{2.8}$$

where x_i is a training example and y_i is its correspondent label.

Concerning the aforementioned methods, batch gradient descent is better suited for smoother curves and converges directly to minima, whereas stochastic gradient descent converges faster for larger datasets. As the stochastic gradient descent can only use one example at a time, it can slow down the overall computations.

Mini-Batch Gradient Descent is a variant that attempts to combine the advantages of both previous approaches. It splits the dataset into smaller batches of size $k << n$, and performs an update for every mini-batch of n training examples. Knowing $x_{i:i+n}$ represents the mini-batches and $y_{i:i+n}$ the labels for each training example within a mini-batch n, we have the following equation:

$$\Theta_{t+1} = \Theta_t - \eta \Delta_\Theta J(\Theta_t, x_{i:i+n}, y_{i:i+n}). \tag{2.9}$$

By using mini-batches, the variance of the parameter updates is reduced, leading to more stable convergence. This approach can also use highly optimized matrix optimizations, making computing the mini-batch gradient very efficient. The parameter k can be adjusted to best meet our data's size, making it adaptable to the data and hardware. The size of the mini-batches usually ranges between 50 and 256.

The three variants all have their advantages as well as disadvantages. Usually, mini-batch gradient descent is more commonly used with NNs. However, other variants could be better suited to different situations or contexts.

2.1.1.3 Gradient Descent Optimization Algorithms

GD algorithms face difficulties, such as selecting an appropriate learning rate or defining the tempo for updating the learning rate. If the learning rate is too low, it can lead to slow convergence, while a learning rate that is too large can retard convergence and cause the loss function to fluctuate around the minimum or even diverge. There are several different optimization algorithms to solve these issues and others.

Momentum (Qian, 1999) is a method that helps accelerate stochastic gradient descent in the relevant direction, allowing it to escape local optima more easily. It also impacts the oscillation provoked in consecutive updates, decreasing the number of up-

dates to converge. This is done by adding a fraction γ of the update vector of the previous time step to the present update vector. Knowing γ is the momentum term, and v_t is the vector of updated values in each iteration, the definition of momentum is:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \Delta_\Theta J\left(\Theta_{t-1}\right), \\ \Theta_t &= \Theta_{t-1} - v_t. \end{aligned} \tag{2.10}$$

Although there are many optimization algorithms in the literature (Ruder, 2017), such as Nesterov accelerated gradient, Adagrad, Adadelta, RMSprop, AdaMax, Nadam or AMSGrad, we will only focus on Adaptive Moment Estimation (ADAM) introduced by Kingma and Ba Kingma and Ba (2015). In brief, ADAM is a combination of momentum (Qian, 1999) and RMSprop (Tieleman et al., 2012), which computes adaptive learning rates for each parameter. It delivers good results alongside sparse gradients or a multitude of parameters. Similarly to momentum, ADAM keeps an exponentially decaying average of past gradients m_t. Being v_t the decaying average of past squared gradients, m_t the average of squared gradients, and β_1 and β_2 momentum terms, the estimates of the first and second moments are:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + \left(1-\beta_1\right) \Delta_\Theta J\left(\Theta_{t-1}\right), \\ v_t &= \beta_2 v_{t-1} + \left(1-\beta_2\right) \left(\Delta_\Theta J\left(\Theta_{t-1}\right)\right)^2. \end{aligned} \tag{2.11}$$

The parameters m_t and v_t are initialized as vectors of zeros, which cause them to be biased towards zero, especially during the initial time steps or when the decay rates are low (i.e., when β_1 and β_2 are close to zero). To counterbalance the biases, bias-corrected first and second-moment estimates are computed:

$$\begin{aligned} \hat{m_t} &= \frac{m_t}{1-\beta_1^t}, \\ \hat{v_t} &= \frac{v_t}{1-\beta_2^t}. \end{aligned} \tag{2.12}$$

Formally, to update the parameters, the Adam update rule is applied using the aforementioned moment estimates:

$$\Theta_{t+1} = \Theta_t - \frac{\eta}{\sqrt{\hat{v_t}} + \epsilon} \hat{m_t}, \tag{2.13}$$

where the parameter ϵ stands for a small scalar to prevent division by zero. Adam works well, achieving better results than other adaptive learning-method algorithms.

2.1.2 The Transformer Architecture

The Transformer (Vaswani et al., 2017) is a recent approach that simplifies tasks related to Neural Architecture sequence-to-sequence processing (i.e., processing sequences of words), which consists of transforming one sequence into another one with the help of two elements: an encoder and a decoder. The model uses an *attention* mechanism that takes into account, as an input, the relationships between all words in an input sentence, providing context for any position in the input sequence.

The Transformer architecture can be seen in detail in Figure 2.2 - The figure shows each of its components, namely the encoder and decoder layers, each attention mechanism, and the Feed-Forward Neural Network (FFNN) blocks. In the following subsections, we will unfold each component.

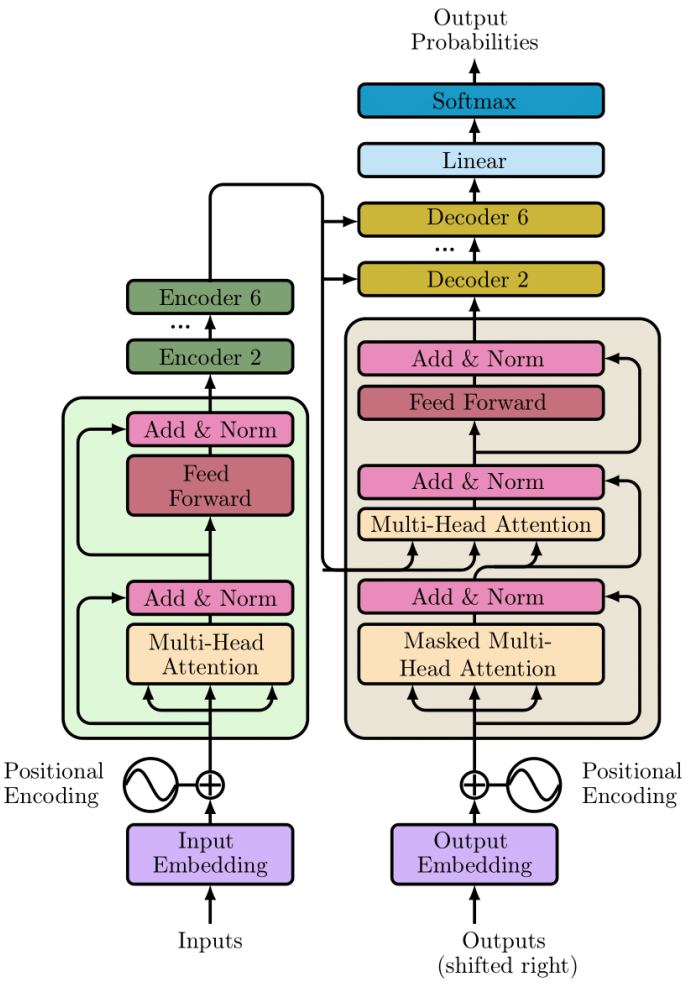


Figure 2.2: The Transformer Model Architecture.

2.1.2.1 Multi-Head Self-Attention

The Transformer uses a scaled *dot-product* attention. This mechanism uses three vectors, respectively called query q, key k, and value v. To obtain them, each vector in the input sequence, which we denote in Tensor form as **X**, has to be multiplied by each vector's weight matrix learnt while training.

$$\begin{aligned} \mathbf{Q} &= \mathbf{X}\mathbf{W}_\mathbf{q}, \\ \mathbf{K} &= \mathbf{X}\mathbf{W}_\mathbf{k}, \\ \mathbf{V} &= \mathbf{X}\mathbf{W}_\mathbf{v}. \end{aligned} \tag{2.14}$$

The role of **Q** is to hold the current words being analyzed, while **K** has an indexing mechanism for the value vectors and **V** saves information about each input word. The next step is to create self-attention for every word in the input sequence. We pick one word in a sentence and calculate the score for all words concerning the chosen word. This result gives the importance of all words to the word in question. The score is given by the *dot-product* of the query vector with each one of the key factors. The closest query-key product will have the highest value, meaning the words associated with the pair are more similar. These scores are then divided by the square root of the dimension key factor to have more stable gradients. To normalize the scores, one applies a *s* function to guarantee they are all positive and add up to 1. The *softmax* function also removes low values by driving them towards low probability scores.

$$\textit{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{2.15}$$

where **z** is a vector with dimensionality K, $i = 1, ..., K$.

The intuition here is to keep intact the values of the words we want to focus on and drown-out irrelevant words by multiplying them by tiny numbers. The idea is to have a **z** vector for each word in a sentence and, in tensor form, we have: finally, each *softmax* is multiplied by the corresponding value vector v. All these results are later summed, forming the output of the self-attention layer, i.e. a **z** vector of size d_k, which will be passed to the Feed-Forward Network (FFN) as an input.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \textit{softmax}\left(\frac{\mathbf{Q}\mathbf{W}^T}{\sqrt{d_k}}\right)\mathbf{V}. \tag{2.16}$$

Each word has an attention vector assigned to itself. However, we need more than one attention vector to capture multiple possible interactions. To solve this, we take eight attention vectors per word and use a weighted average to compute the final attention vector for every word. Since the self-attention operation is calculated multiple times, in parallel and independently, this process is commonly known as multi-head attention.

The main advantage of multi-head attention is training stability since this way, we can use a smaller number of layers than single-head attention. Attending the same number of positions gives the model a greater power to encode multiple relationships and nuances for each word. Using h parallel attention layers, commonly known as heads, the multi-head attention mechanism allows the model to attend to information from various representation subspaces concurrently. To do so, we use distinct projected **Q**, **K**, and **V** values in each head to calculate attention. Consequently, we combine the findings and project the resulting concatenation into the original dimensions.

Formally, this may be written as follows:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}\left(head_1, head_2, ..., head_h\right)\mathbf{W}^O, \tag{2.17}$$

where

$$\text{head}_i = \text{Attention}\left(\mathbf{Q}\mathbf{W}_i^{\mathbf{Q}}, \mathbf{K}\mathbf{W}_i^{\mathbf{K}}, \mathbf{V}\mathbf{W}_i^{\mathbf{V}}\right). \tag{2.18}$$

The letter h stands for the number of existing attention heads, whereas the letters $\mathbf{W}_i^{\mathbf{Q}}$, $\mathbf{W}_i^{\mathbf{K}}$, $\mathbf{W}_i^{\mathbf{V}}$ and $\mathbf{W}^O$ are parameter matrices.

2.1.2.2 Input for the Model

Before the inputs go inside the encoder/decoder layers to be processed, we have to change their representation using embeddings and positional encoding. Word embeddings are word representations that allow words with similar meanings to have exact representations, capturing their meaning. They help capture the semantics or context and help to understand the relationship between terms in the sentence. To generate embeddings, we apply feature extraction-based techniques to map words or phrases from the vocabulary to vectors of real numbers, which can be later used in computations. For the Transformer, each token of our input sequence will be transformed into a vector of size $d = 512$. This size is kept constant throughout all steps of the model. A word embedding layer can be explained as a table where the model grabs a learned

vector representation for each word of a phrase.

The self-attention mechanism does not consider the order of the input sequence, which is an essential aspect of many NLP tasks. Therefore, there is a need to implement a mechanism that attributes to each token a relative position. This is achieved by adding a position encoding vector to each input embedding. This is only done before the word embeddings are fed into the model. The encoding vectors are not learnt as part of the model; instead, they are hard-coded using the following function:

$$\begin{aligned} \mathsf{PE}(pos, 2i) &= \sin\left(\frac{pos}{1000^{\frac{2i}{d}}}\right), \\ \mathsf{PE}(pos, 2i+1) &= \cos\left(\frac{pos}{1000^{\frac{2i}{d}}}\right), \end{aligned} \tag{2.19}$$

where *pos* is the token's position in the text sequence and *i* the vector dimension, so that even and odd dimensions get different positional encodings. Each dimension of the position encoding corresponds to a sinusoid.

The embedding and position encoding of the input sentence make every token unique. Two properties encoders inform the model: the absolute position of each word in the input sentence and the relative distance between different words.

2.1.2.3 The Encoder Component

In a Transformer, the encoder is formed by six smaller sequential encoder layers, each containing the same structure but different parameters. Each encoder contains two sub-layers, namely a multi-head attention layer whose goal is understanding how neighbour positions affect the current one, followed by a fully-connected FFNN layer, sharing the same input and output formats.

The position-wise Feed-Forward Networks (FFN) is present in both encoder and decoder layers after each multi-head attention sub-layer. This NN is not incorporating information from other words. It looks at only one word simultaneously, contrasting with the attention layer. This consists of two linear transformations with a RELU activation among them. The input and output layers have size $d_{model} = 512$, whilst the inner layer has 4 times the size of the model $d_{ff} = 2048$.

There is also a residual connection (He et al., 2016) around each sub-layer and a normalization operation that normalizes each layer's result. Assuming x_1 is one vector of embeddings received by an encoder, and x_3 is the output of one of the six smaller

encoders, the normalization operation can be formally expressed as follows:

$$\begin{aligned} x_2 &= \text{LayerNorm}\left(x_1 + \text{MultiHead}(x_1)\right), \\ x_3 &= \text{LayerNorm}\left(x_2 + \text{FFNN}(x_2)\right). \end{aligned} \tag{2.20}$$

The multi-head attention is explained in Section 2.1.2.1, whereas the normalized layer adds the input x to the implemented sub-layer function output. The way the LayerNorm (Ba et al., 2016) operator is applied to the embeddings ν is formulated as:

$$\text{LayerNorm}(\nu) = \gamma \frac{\nu - \mu}{\sigma} + \beta, \tag{2.21}$$

where μ and σ are the mean and standard deviation of the elements in ν, and γ, β are hyperparameters associated with scale and bias vector. The output of the encoder is the following small encoder and the multi-head attention layer of the decoder.

2.1.2.4 The Decoder Component

The decoder is composed of a stack of 6 identical layers, similar to the encoder, and each decoder layer has a similar structure to each encoder layer. The main difference between the two is the addition of a masked multi-head attention layer in the beginning. The decoder's inputs are embedded, and the position is encoded the same way as the encoder's input. The resulting vector of the embedding and encoding is fed into the masked-headed self-attention sub-layer. The term *masked* in this type of attention symbolizes that the predictions for position *j* can only depend on the known outputs of positions less than *j*.

The non-masked attention layer of the decoder, also known as the encoder-decoder attention layer, receives inputs from either the decoder's masked layer or the encoder's output. This configuration helps the decoder focus on appropriate parts of the input sequence. The encoder-decoder's output sequence prediction is achieved autoregressively, which means it is created iteratively, token by token. The output of the decoder stack is then fed back into the following decoder. Assuming x_1 is an embeddings vector, the architecture of a decoder is defined as follows:

$$\begin{aligned} x_2 &= \text{LayerNorm}\left(x_1 + \text{MaskedMultiHead}(x_1)\right), \\ x_3 &= \text{LayerNorm}\left(x_2 + \text{EncoderDecoderMultiHead}(x_2)\right), \\ x_4 &= \text{LayerNorm}\left(x_3 + \text{FFNN}(x_3)\right). \end{aligned} \tag{2.22}$$

The output embedding vector (x_4) is then passed to a linear layer containing the output dimensionality of the vocabulary. Finally, the last operation is a *softmax* function, which computes the predicted probabilities for each word in the chosen vocabulary.

2.1.3 Bidirectional Encoder Representation from Transformers

Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019a) is a language model developed from the Transformer model. BERT was a disruptive technology, as it had impressive benchmark performances in different NLP tasks. The main idea is to use a Transformer encoder to produce deep bidirectional representations from the text that can be explored for target tasks. BERT produces contextual token representations because the same token can have a different representation vector depending on the surrounding context.

BERT is a PLM, meaning it can be chosen from available models, where they are pre-trained over a large unlabeled generic corpus to perform a specific task. If suitable, the model can be used as the starting point for a model on the task of interest. However, one might wish to adapt a particular task or dataset over additional data; this is fine-tuning. From this approach, we can effectively adapt the model with generic language understanding capabilities to understand our dataset (i.e., apple may have two meanings: a fruit or a technology brand). BERT uses two pre-training tasks, namely Masked Language Model (MLM), and Next Sentence Prediction (NSP).

In the first task, displayed in Figure 2.3, we randomly mask a percentage of the input tokens (i.e., *fat, eat*), and try to predict them. In Figure 2.3, the masked tokens are fed into an output *softmax* over the vocabulary. Such operation enables us to obtain a bidirectional pre-trained model. However, there is a limitation. If we use this task for fine-tuning, the [MASK] token is not visible. A possible fix is not to assign a [MASK] token to every *masked* word, meaning only a small percentage of token positions are randomly assigned to be predicted, meaning each token has a defined probability of being either replaced by the [MASK] token, to be replaced by a random token, or not to be replaced. Finally, the final hidden vector for the token to be predicted is used to predict the original token with cross-entropy loss.

The second task displayed in Figure 2.3 is used for the model to understand the relationship between two sentences. The task consists of classifying whether sentence A is followed by sentence B. Each sentence has a label designated *isNext* or *notNext*. We assign sentence B as the actual next sentence 50% of the times when performing

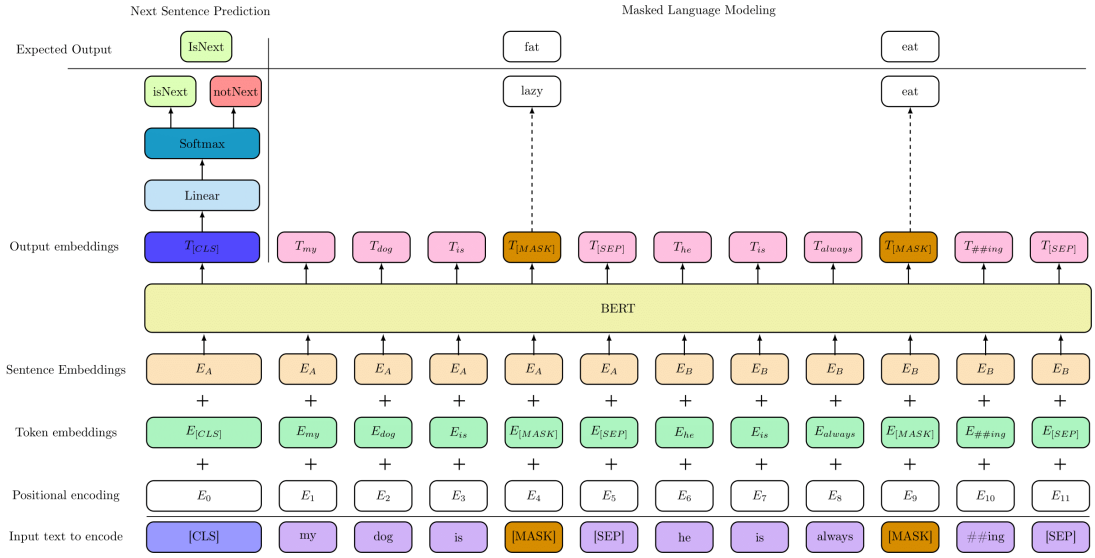


Figure 2.3: Input Representation and Pre-Training tasks from BERT.

the task. In the remaining attempts, the *notNext* labelled sentence is randomly picked from the corpus. We compare the predicted output with the expected output and denote if their labels (*isNext*, *notNext*) match. Pre-training towards this task is very beneficial to Question-answering tasks.

After pre-training the model, we can modify BERT to perform certain tasks by replacing its output layers. In the original BERT implementation (Devlin et al., 2019a), the pre-training was done using the BooksCorpus (800 million words) and English Wikipedia (2500 million words) as datasets.

2.1.3.1 BERT Inputs

In the context of BERT, an input sequence is defined as an arbitrary span of contiguous text rather than a linguistic sentence. The BERT input representation is developed to represent one or more sentences as a token sequence. The sequence of tokens has to go through some pre-processing steps before being fed into BERT: token embeddings (1), sentence embeddings, and (2) positional embeddings (3).

Every word token in the input sequence has to exist in BERT's fixed vocabulary, so every word has to be converted through the token word piece embedding (Zhang et al., 2019) into a vector representation, as it can be seen in Figure 2.3. In the word embedding vocabulary lookup table, about 30,000 words and 768 features correspond to the output vectors of BERT. Each of the tokens can define a word, a set of characters, a symbol or a special mark. One example of a special mark is the [CLS] token, which is placed at the start of every input sentence, whereas [SEP] is used to separate

segments of a sequence when there is more than one sentence (e.g., [SEP] separates the sentences “my dog is fat” and “he is always eating”). Furthermore, if a word does not exist in the vocabulary, it is split into smaller subwords. All subwords start with ## except for the first one (i.e., if the word eating does not exist in the vocabulary, it is tokenized to eat + ##ing). All sentences must be truncated to a single fixed length, and the maximum allowed length is 512 tokens.

The token embeddings are then added to a segment embedding that identifies which sentence a token belongs to within a given sequence. If only one sentence exists, this embedding will be the same for every token.

A position embedding is also added. In the Transformer architecture (Vaswani et al., 2017), the position is encoded through a hard-coded function. However, BERT model uses learned embeddings, trained with sequence length 128 for 90% of the steps, and later remaining 10% of steps were trained with a sequence length of 512 tokens to learn position embeddings. These embeddings are applied to determine the global position of any given token and the relative distance among tokens. For every token, its input representation is thus obtained by summing the corresponding token, segment, and position embeddings. All output vectors resulting from the embedding layer are unique.

2.1.3.2 The Overall BERT Architecture

The base implementation of BERT has an embedding layer followed by twelve modified Transformer encoder blocks. The size of the embedding vector was increased by 50$\%$ $(512 + 256)$ to 768, and the number of attention heads increased to 12. The mechanism of each encoder sub-layer is the same as described in Section 2.1.2.3. The self-attention layer is shared among all tokens.

2.1.3.3 BERT Fine-Tuning

One of BERT’s most significant advantages is adapting the model to multiple applications. The same model can be applied to multiple downstream tasks involving single texts or text pairs by adjusting the adequate inputs and outputs. As the number of parameters of BERT can range from 100 million to over 300 million, training the model from scratch on a small dataset would result in overfitting. We can consider three different fine-tuning techniques:

1. Train the entire architecture: the error is back-propagated through the entire architecture, and the pre-trained weights are updated based on the new dataset.

2. Train some layers while freezing others: keep the weights of the initial layers of the model frozen while we re-train only the higher layers. We can manually assign how many layers we wish to be frozen and how many to be trained.

3. Freeze the entire architecture: we can freeze all the layers of the central architecture and attach NN layers to train the new model. Only the weights of the attached layers would be updated during model training.

BERT by itself, we cannot perform the majority of tasks; therefore, we have to adapt the model for the task we want to evaluate. If we wish to do token-level tasks, we can feed the output token representations to the final classification part of the model (i.e., a *softmax* layer). On the other hand, we can provide the [CLS] token representation to a final classification head for sentence-level tasks.

2.1.3.4 BERT Variations

Recently, several researchers have proposed extensions of BERT with different configurations that deliver better performance in multiple areas. An example is Robustly optimized Bidirectional Encoder Representations from Transformers (RoBERTa) (Liu et al., 2019), which was one of the first proposals. The main idea was to improve BERT pre-training. Unlike BERT, RoBERTa uses dynamic masking over the training data to a different masking pattern every time a sequence is fed into the model. Furthermore, the authors also suggested that the original model was undertrained, and by altering some aspects, state-of-the-art results could be matched or surpassed in several datasets. Some alterations were made, such as increasing the number of training steps, training the model with larger batches over more data, removing the next sentence prediction task, and training on longer sequences.

Improving Pre-training by Representing and Predicting Spans (SpanBERT) (Joshi et al., 2020) aims to improve the representation and prediction of text spans in BERT. The authors proposed (1) masking random spans instead of random tokens and (2) training the span boundary representations to predict the entire content of the masked span without relying on individual token representations. The Span-based masking enables the model to predict entire spans using only their context. This model outperformed BERT in all baselines, mainly when tasks involve contiguous spans of text.

Efficiently Learning and Encoder that Classifies Token Replacements Accurately (ELECTRA) (Clark et al., 2020) introduced a new training approach that produces a

model with improved performance compared to the best transformer implementation, requiring only a fraction of the computational power. The author noted that the Masked Language Modeling task requires a large amount of data to work correctly. They proposed a more sample-efficient pre-training task called *replaced token detection*. Rather than predicting the masked token based on its neighbours, the model is tasked with unique and plausible input tokens created by a small generator network (i.e., a smaller MLM). This method enables all input tokens to train the model, rather than just the masked token, explaining the efficiency of ELECTRA. Instead of training the model to identify the original identity of the masked tokens, the approach is to predict token by token, whether original or generated. As the training is being done in input tokens, we have access to a much more extensive training set with the same input data.

The general solution to produce a multilingual BERT is to pre-train on a mixture of many languages by learning knowledge shared between idioms and leveraging it for fine-tuning tasks of any language contained in the mixture. mBERT (Devlin et al., 2019b) is an adaptation of BERT to languages other than English, simultaneously supporting 104 languages. For every language, all the text is lowercase, the diacritics are removed, the punctuation is split, and all the whitespaces are tokenized.

Decoding-enhanced Bidirectional Encoder Representations from Transformers with Disentangled Attention (DeBERTa) (He et al., 2021) is another Transformer-based neural language model pre-trained on large amounts of raw text using self-supervised learning. By introducing two main innovations, the architecture improves BERT and RoBERTa. The first idea is a disentangled attention mechanism that splits the input vector into two vectors: content and position. The attention weights are then computed using disentangled matrices over each one separately. The second innovation was incorporating absolute positions in the Masked Language Modeling task by applying word position embeddings before the vectors enter the *softmax* layer. Furthermore, the author introduced a new virtual adversarial training method for fine-tuning the model to NLP tasks, which proved effective at improving model generalization.

A Lite Bidirectional Encoder Representations for Self- supervised Learning of Language Representations (ALBERT) (Lan et al., 2020) introduced two parameter-reduction techniques to lower memory consumption and increase the training speed of BERT. The first one is a factorized embedding parameterization. The large vocabulary embedding matrix is decomposed into two smaller matrices, and the size of the hidden

layers is separated from the size of the vocabulary embeddings. This alteration makes it easier to grow the hidden size without increasing the parameter size of the vocabulary embeddings. The second innovation corresponds to cross-layer parameter sharing. This prevents the parameters from growing with the depth of the network. By applying these techniques, the number of parameters for BERT was reduced, and the performance was not compromised. Comparing an ALBERT configuration to a similar BERT-large model, the first one has about eighteen times fewer parameters and can be trained about 1.7 times faster. The parameter reduction also stabilizes the training and helps with generalization.

2.1.3.5 Important Limitations in BERT

Experiments with different NLP tasks have shown that BERT implementations struggle with role-based event prediction and that they show apparent failures with the meaning of negation (Ettinger, 2020). It does not have sensitivity to the most probable answer in real-life situations. Also, BERT cannot deal with long input sequences. BERT only supports up to 512 tokens, and since the self-attention mechanism has quadratic computational and memory requirements, changing the supported token limit would introduce a significant increase in computational cost.

BERT also highly depends on the training dataset vocabulary. Suppose the model performs tasks over dynamic areas of knowledge where new words or types of writing are added from time to time. In that case, it will not recognize the unlabeled terms and will have a more data-scarce vocabulary in that field. A solution to deal with this is to perform continuous re-training and validation, but on top of being time-consuming, this is also costly and not practical.

2.2 Related Work

NLP involves tasks regarding entities, such as: Entity Typing (ET), Named Entity Recognition (NER), Question Answering (QA) and Relation Classification (RC), just to name a few. A good representation of entities is a decisive part of a good model.

Although transformer-based representations, e.g., produced by BERT (Devlin et al., 2019a) and RoBERTa (Liu et al., 2019), correspond to good word representations, they are not very effective in representing entities because named entities are “open” linguistic constructs, meaning they are difficult to model (i.e., it is not possible to have

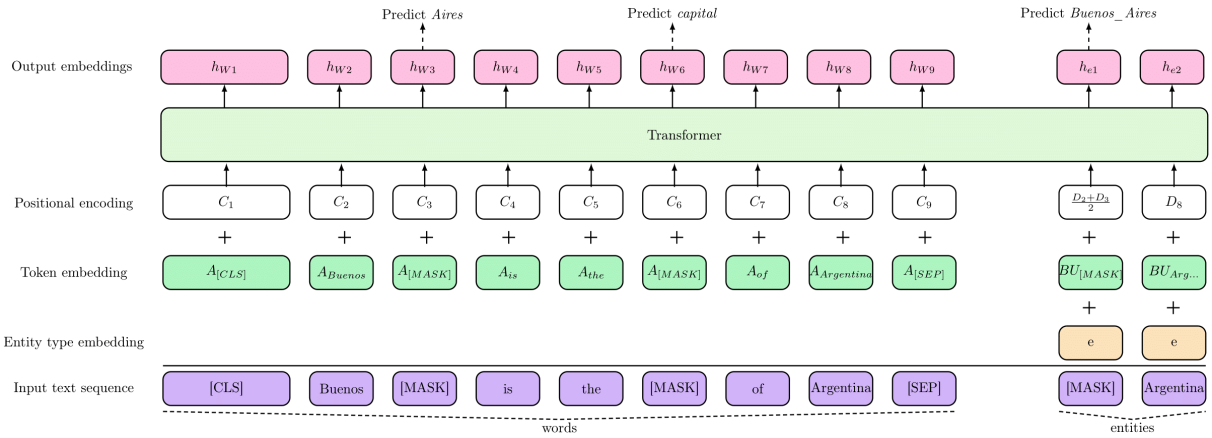


Figure 2.4: Input Representation from LUKE.

all existing names of people in a training dataset). Even though the Transformer self-attention mechanism captures complex relationships between words, it cannot establish the relationship between entities when they are composed of more than one word. Although recent approaches (Qin et al., 2021a) proposed to improve entity representations by aiming to obtain a deep understanding of the entities and their relations in text, most models need help to model entity representations effectively.

2.2.1 Language Understanding with Knowledge-based Embeddings

Language Understanding with Knowledge-based Embeddings (LUKE) (Yamada et al., 2020) is a language model based on RoBERTa (Liu et al., 2019) designed to improve the performance of entity-related tasks. This approach adds entity embeddings as well as a new entity-aware self-attention mechanism. LUKE learns how to compute entity representations and deliver them in the model output.

2.2.1.1 Pre-training Task

The model is pre-trained through a Masked Language Model (MLM) task and a new pre-training task that focuses on learning dedicated entity-based representations. This is depicted in Figure 2.4, where it is possible to observe that the right side of the model's architecture focuses specifically on entities. The model learns these representations by targeting entities on a separate MLM task. Note that LUKE encapsulates the entirety of entities such as "Buenos Aires" or "Lord of the Rings" in the same token to capture less predictable entities better. To this end, the model must predict 15% of all words and entities during training. If an entity is not represented in LUKE vocabulary, the [UNK] token is assigned to the referred entity. Finally, we predict the masked entity y.

The dataset used to train the novel task is extracted from a large amount of Wikipedia data, where the entities are the hyperlinks present on the Wikipedia website. The prediction of the masked entity $\hat{y}$ is formulated as follows:

$$\begin{aligned} m &= \text{Layer\_Norm}\left(\text{gelu}(W_h h_e + b_h)\right), \\ \hat{y} &= \textit{softmax}(BTm + b_0), \end{aligned} \tag{2.23}$$

assuming h_e is the Transformer layer output associated with the masked entity, T and W_h correspond to weight matrices, *softmax* and Layer_Norm correspond to the activation and layer normalization functions, respectively, B represents the entity token embedding matrix, and m stands for the number of words. The output of LUKE is a contextualized representation for each word and entity in the text.

2.2.1.2 Input Representation

A unique attribute of LUKE is the treatment of words and entities as independent tokens, representing each one, as seen in Figure 2.4. The input representation of a token is obtained by applying the following embeddings:

1. Token embedding: represents the type of token(i.e., A for words or B for entities);

2. Positional embedding: represents the token's absolute position in a word sequence. If an entity includes more than one token, its position embedding D is calculated by averaging the embeddings of the corresponding token positions C that form the entity;

3. Entity type embedding (e): means that the token is an entity.

The input representation of a word (i.e., left side of the input sequence in Figure 2.4) is given by adding the token and position embeddings, whereas to obtain the entity representation (i.e., *Buenos Aires*$_{[MASK]}$, *Argentina*) we add all the embeddings. As the masked input entity relates to multiple token entities (*i.e., Buenos Aires*), we average the corresponding position embeddings to encode its position. For example, the entity *Buenos Aires* occupies positions 2 and 3 of the input word sequence; therefore, to obtain the positional entity embedding, we have $\frac{D_2+D_3}{2}$. Similarly to previous model implementations (Devlin et al., 2019a; Liu et al., 2019), LUKE makes use of special tokens [CLS] and [SEP] to mark the first and last words of an input sequence.

2.2.1.3 Entity-Aware Self-Attention Mechanism

The idea behind the standard self-attention mechanism is to allow tokens to relate using attention scores. This is also implemented in LUKE with a few modifications. The traditional mechanism (Equation 2.16) does not expect two different kinds of tokens, while LUKE proposes a new attention mechanism that is aware of what kind of tokens it is dealing with when it is computing the attention score (e_{ij}) between the pair of tokens. The idea mentioned above was implemented through an entity-aware query mechanism that handles query matrices with different weights for each possible combination of pairs of tokens x_i and x_j.

The proposed attention mechanism is computed as follows:

$$e_{ij} = \begin{cases} \mathbf{K}x_j^T\mathbf{Q}x_i \text{ , if } x_i \text{ and } x_j \text{ are words} \\ \mathbf{K}x_j^T\mathbf{Q}_{w2e}x_i \text{ , if } x_i \text{ words and } x_j \text{ entity} \\ \mathbf{K}x_j^T\mathbf{Q}_{e2w}x_i \text{ , if } x_j \text{ words and } x_i \text{ entity} \\ \mathbf{K}x_j^T\mathbf{Q}_{e2e}x_i \text{ , if } x_i \text{ and } x_j \text{ are entities} \end{cases}, \tag{2.24}$$

where **Q**, $\mathbf{Q}_{w2e}$, $\mathbf{Q}_{e2w}$ and $\mathbf{Q}_{e2e}$ are query matrices, and where **K** and **V** are, respectively, the key and value matrices.

2.2.1.4 Experiments

The model delivered state-of-the-art performances in 5 entity-related tasks with only one simple linear layer after the token representations.

- Entity typing: the goal is to predict the type of each entity (i.e., person, location, organization) in a given sentence.
- Relation classification: predict a relation between *head* and *tail* entities in a given sentence.
- Entity recognition: locate and classify named entities mentioned in unstructured text into pre-defined categories.
- Cloze-style QA: given a question and a passage of a text, identify the suitable entity.
- Extractive QA: predict the correct answer to a given question when a Wikipedia excerpt containing the answer is provided.

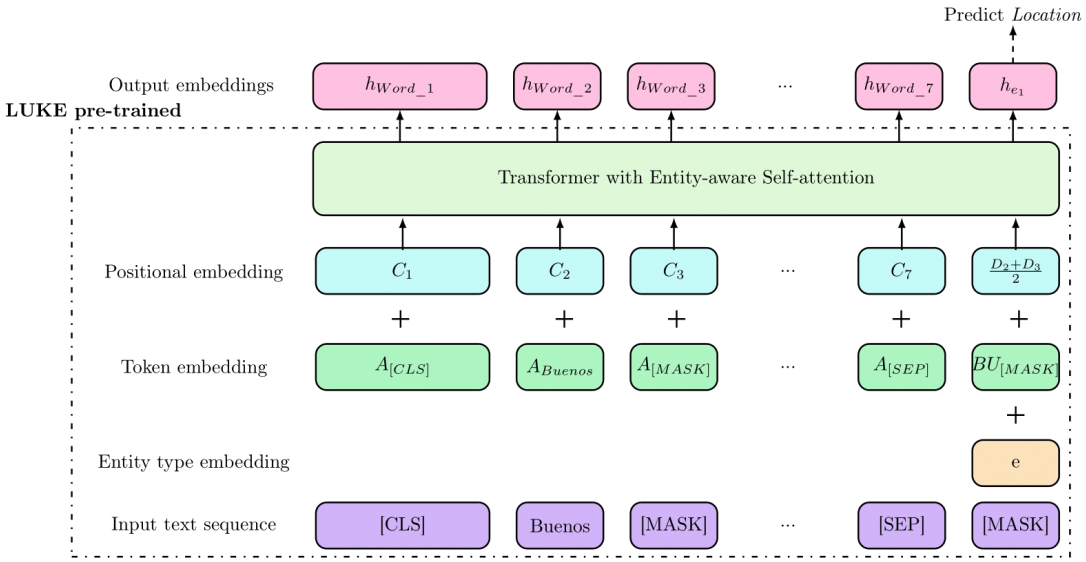


Figure 2.5: Entity typing task from LUKE.

In Figure 2.5, we can see an example of how we can apply the representations LUKE gives us on a NLP task, particularly on the entity typing task. In this example, we mask the target entity and insert words and entities in each sentence as the model's inputs. The target entity is classified using a linear classifier based on the corresponding entity representation. Like a multi-label classification task, the model is trained using binary cross-entropy loss averaged over all eight possible entity types to predict which type corresponds to the target entity. In the case of Figure 2.5, the entity type of the entity *Buenos Aires* is *Location*. LUKE is effective at capturing relationships between entities. The introduction of entity-aware self-attention, and a personalized entity representation, enabled the model to outperform RoBERTa on every entity-related task.

2.2.1.5 Limitations

Although LUKE outperformed some models in important tasks, it also encountered some drawbacks. LUKE delivers good performance when the model receives both word and entity-type tokens. As LUKE highly depends on datasets that label their entities, if a new entity that does not belong in the vocabulary is introduced, the model has to fall back to the [UNK] entity representation, damaging its performance in the target NLP task. However, possible solutions could be to apply techniques referenced in Improving Entity and Relation Understanding for Pre-trained Language Models via Contrastive Learning (ERICA) (Qin et al., 2021b).

Another LUKE limitation is the inability to handle long sequences of text as its maximum sequence length is only 512 tokens. This is due to the self-attention operation, which scales quadratically with the sequence length. A possible solution for this issue would be to borrow technical ideas from Longformer (Beltagy et al., 2020),Transformers for Longer Sequences (BIG BIRD) (Zaheer et al., 2020), and Rabe and Staats (2021) to reduce the quadratic dependency to linear.

Although LUKE was designed for English text, there is an adapted version of this model named m-LUKE (Ri et al., 2021), supporting 24 different languages. One of the proposed ideas is to match Wikipedia hyperlinks that relate to the same article (i.e., *Lisbon* and *Lisboa* are different entity annotations. However, they share the same entity and Wikipedia article). The detected entity tokens that share the same entities with the input sentence are stacked to the input sequence of the model. This way, entity tokens are expected to provide the model with language-independent features.

2.2.2 ERICA

A limitation with most previous Transformer-based models is the inability to explicitly model relational facts in the text, which is a significant for textual understanding. Even though some studies have tried to find solutions for this obstacle (Baldini Soares et al., 2019), they only concern individual entities or relations between pairs of entities within the same sentence. The interactions among more distant and complex entities are entirely ignored. To obtain a deeper understanding of entities and their relations in text, Qin et al. (2021b) proposed ERICA, introducing two new tasks for enabling the model to deal with entities and their relations.

Specifically, the model is trained using the information of knowledge graph $\mathcal{K}$ to learn all existing entities and relations. For each document d_i belonging to a batch of documents $\mathcal{D}$, all entity pairs (e_{ij}, e_{ik}) are annotated and linked to their possible relation r^i_{jk} present in $\mathcal{K}$. If a pair has no relation annotated in $\mathcal{K}$, it obtains the *no_relation* status. Each document contains information regarding the relations between entity pairs in a tuple. Then, the tuples for each document are united to form an overall tuple $\mathcal{T}$. The set of tuples excluding the *no_relation* ones define the set $\mathcal{T}^+$. A regular Transformer-based language model encodes each document d_i and outputs a hidden state for each token. To the series of hidden states $\{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_{d_i}\}$, we apply *mean pooling* over consecutive tokens that mention the j^{th} word of the i^{th} document to obtain local entity representations. Using *mean pooling* to encode entities helps the model

to represent unknown entities. The index k stands for the number of occurrences of the entity in the document d_i. Formally, the k^{th} occurrence of e_{ij} corresponds to the following equation:

$$\mathbf{m}_{e_{ij}}^{k} = \text{MeanPool}\left(\mathbf{h}_{n_{\text{start}}^{k}}, ..., \mathbf{h}_{n_{\text{end}}^{k}}\right), \tag{2.25}$$

where n_{start}^{k}, and n_{end}^{k} are indexes. In order to condense information about each unique entity, we set the global entity representation e_{ij} as the arithmetic average of all representations of each occurrence $m_{e_{ij}}^{k}$. Afterwards, we concatenate the final representations of two entities e_{ij1} and e_{ij2} as their relation representation $r_{j_1 j_2}^{i}$. ERICA introduced two new pre-training tasks:

- Entity discrimination task: Considering a given *head* entity and the relation between the pair of entities, the objective of this task is to predict the *tail* entity in a document text. The prediction is then compared to the ground-truth *tail* entity corresponding to the correct relation. This task motivates ERICA to understand an entity by considering its relation with the remaining entities.
- Relation discrimination task: This task aims to pick two entities and verify whether they semantically relate. In this specific task, the document's targeted entities are distant to comprehend complex relations in real-world scenarios better.

Along with the two novel tasks, the model uses MLM during training for the model not to lose the ability to understand general language (McCloskey and Cohen, 1989). Formally, the overall learning objective ($\mathcal{L}$) of ERICA is formulated as:

$$\mathcal{L} = \mathcal{L}_{ED} + \mathcal{L}_{RD} + \mathcal{L}_{MLM}. \tag{2.26}$$

ERICA was benchmarked with the BERT and RoBERTa models in document-level Relation Extraction (RE), ET, and QA, outperforming all the established baselines. This shows the effectiveness of ERICA in leveraging relations between pairs of entities to learn entity-based representations better. The concepts mentioned earlier in this Section resolve one of the obstacles of LUKE.

However, ERICA shares with LUKE the same limitation regarding handling longer sequences of text at a time. Also, to improve entity representation, ERICA could benefit from the addition or replacement of different methods for entity encoding (i.e., pairs of marker tokens $[S_i], [E_i]$ that denote the limits of each representation of an entity in the document) (Baldini Soares et al., 2019).

2.3 Chapter Overview

Section 2.1 of this Chapter expands upon the structural knowledge required to understand the exposed methods, allowing Section 2.2 to focus on providing a broad overview of what the state-of-the-art entity-related models are, their architecture, and the influence they had on future work. A particular emphasis was deposited on LUKE and leveraging entities as this work's primary focus.

Chapter 3

The Proposed Approach

Following recent trends in NLP, we present Long-LUKE. This new entity-based architecture combines the concepts of using complex entity representations from LUKE and modifying the attention pattern (Beltagy et al., 2020) to enhance the model's performance on long documents. This model aims to solve the limitations of LUKE by converting the original model into a Long-Transformer. Although LUKE outperformed some models in essential tasks, it also encountered some drawbacks. As LUKE highly depends on datasets that label their entities, if a new entity that does not belong in the vocabulary is introduced, the model has to fall back to the [UNK] entity representation, damaging its performance in the target NLP task.

Another LUKE limitation is the inability to handle long sequences of text as its max_seq length is only 512 tokens due to the self-attention operation, which scales quadratically with the sequence length. Recently, other studies proposed similar methods to address this limitation (Zaheer et al., 2020; Rabe and Staats, 2021; Jiang et al., 2020; Kitaev et al., 2020). The solution we developed borrows technical ideas from Longformer (Beltagy et al., 2020) to reduce the quadratic dependency to linear.

3.1 The Long-Document Transformer

An example of a Long-Transformer is Longformer (Beltagy et al., 2020) which contains an approach to reduce the time and memory complexity of the full self-attention mechanism (Figure 3.2(a)). It proposes to design a new self-attention pattern composed of several elements that define the pairs of tokens matched in the attention calculation. The **sliding window attention pattern** (Figure 3.2(b)) applies a fixed-size window surrounding each token. Several of the aforementioned windowed attention

layers are stacked to generate a broad receptive field. Each token attends to $\frac{1}{2}w$ tokens on each side, where w stands for the defined window size. This pattern causes $\mathcal{O}(n \times w)$ complexity, which is linear to the input sequence length n. Subsequently, Longformer introduces size dilation gaps d to the window (Figure 3.2(c)). In this implementation, the authors varied the size of the windows across the layers. Notably, small window sizes are used for lower layers. The window size increases as we move to higher layers, enabling the top layers to learn a higher-level representation of the entire sequence while having the lower layers capture local information. Additionally, it also provides a balance between efficiency and performance. Lower layers do not use dilated sliding windows to maximize their capacity to learn the local context.

The **global attention pattern** (Figure 3.2(d)) is added on pre-defined input locations in order to learn task-specific representations, such as [CLS], [UNK], and [SEP]. In this pattern, the attention operation is symmetric, meaning a token with global attention attends to all tokens across the sequence and vice versa. The number of tokens covered by global attention is small relative to n. Therefore the complexity of the combined attention patterns is still $\mathcal{O}(n)$. The **combined attention pattern** is shown in Figure 3.2(d). It is possible to observe that it corresponds to the combination of the previous patterns. Converting a Transformer into a Long-Transformer is still effective even when applied only during fine-tuning, which avoids re-training the PLM from scratch.

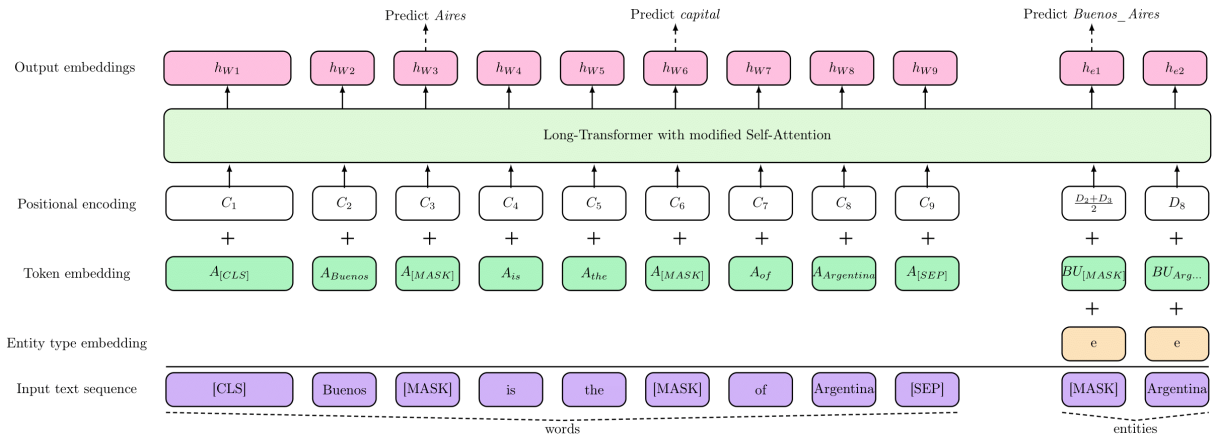
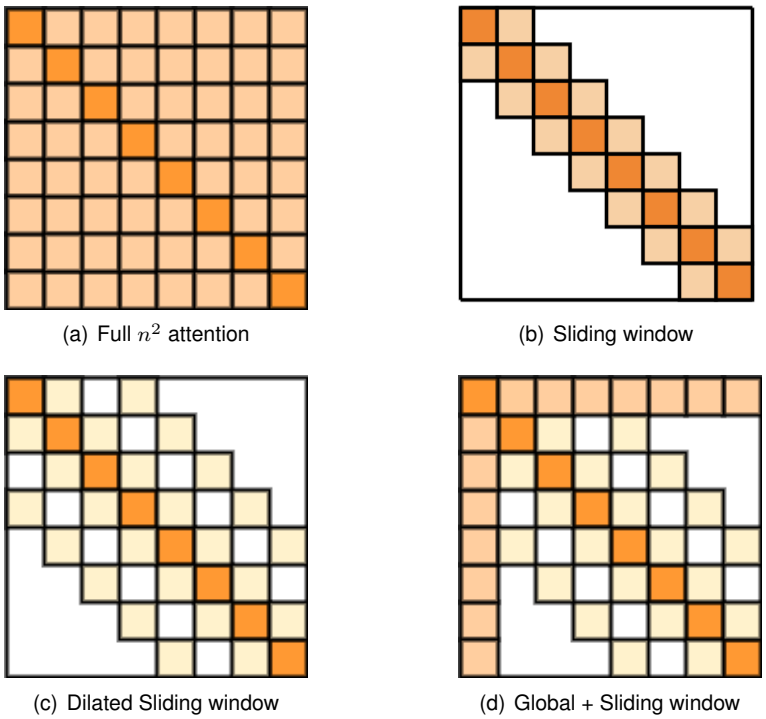


Figure 3.1: Representation of the Architecture of Long-LUKE

3.2 Details on Long-LUKE

For the most part, our model maintains the same architecture of the original LUKE implementation. Particularly, Long-LUKE has both a Transformer (Vaswani et al., 2017) component from BERT, since it treats both words and entities as independent tokens, and Long-Transformer (Figure 3.1) characteristics because it unlocks the ability of the model to deal with longer documents without much memory effort. The primary adjustment is using a custom self-attention mechanism inspired by both Vaswani et al. (2017) and Beltagy et al. (2020) that deals with both word and entity tokens.



(a) Full n^2 attention (b) Sliding window

(c) Dilated Sliding window (d) Global + Sliding window

Figure 3.2: Long-Transformer configuration of attention patterns.

3.2.1 Adapting the Input Representations

A unique attribute that Long-LUKE shares with LUKE is the treatment of words and entities as independent tokens. The input representation of a token, which is either a word or an entity, is obtained by applying the following embeddings:

1. Token embedding: represents the type of token(i.e., A for words or B for entities);

2. Positional embedding: represents the token's absolute position in a word sequence. If an entity includes more than one token, its position embedding D is

calculated by averaging the embeddings of the corresponding token positions C that form the entity;

3. Entity type embedding (e): means that the token is an entity.

The input representation of a word (i.e., left side of the input sequence in Figure 3.1) is given by adding the token and position embeddings, whereas to obtain the entity representation (i.e., *Buenos Aires*$_{[MASK]}$, *Argentina*) we tokenize the entire span of the entity as one, add tokens to position tokens and entities to position entities. As the masked input entity relates to multiple token entities (*i.e., Buenos Aires*), we average the corresponding position embeddings to encode its position. For example, the entity *Buenos Aires* occupies positions 2 and 3 of the input word sequence. Therefore, to obtain the positional entity embedding, we have $\frac{D_2+D_3}{2}$.

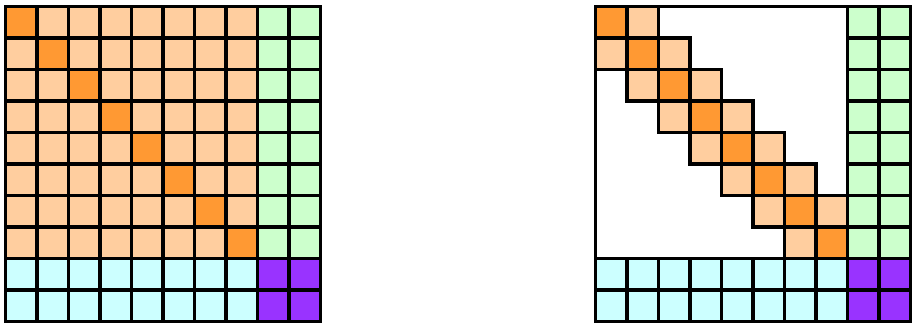
3.2.2 Adapting the Self-Attention Implementation

A key contribution of Long-LUKE is that it extends LUKE using our *Long entity-aware* self-attention mechanism by combining the ideas behind the self-attention patterns described by Vaswani et al. (2017) and Beltagy et al. (2020). As our model is aware of what kind of tokens it is dealing with when computing the attention score (e_{ij}) between the pairs of x_i and x_j tokens, we developed our attention mechanism such that the attention scores where both tokens are words (e_{ww}) are computed through Longformer methodology, maintaining the LUKE operations for the remaining cases (e_{we}, e_{ew}, and e_{ee}). The attention pattern of $w2w$, $w2e$, $e2w$, and $e2e$ are illustrated in Figure 3.2 by the color scheme. Formally, the attention mechanism is computed as follows:

$$\mathsf{e}_{ij} = \begin{cases} \text{Sliding Window Attention , if } x_i \text{ and } x_j \text{ are words} \\ \mathbf{K}x_j^T\mathbf{Q}_{w2e}x_i \text{ , if } x_i \text{ words and } x_j \text{ entity} \\ \mathbf{K}x_j^T\mathbf{Q}_{e2w}x_i \text{ , if } x_j \text{ words and } x_i \text{ entity} \\ \mathbf{K}x_j^T\mathbf{Q}_{e2e}x_i \text{ , if } x_i \text{ and } x_j \text{ are entities} \end{cases} , \tag{3.1}$$

where $\mathbf{Q}$, $\mathbf{Q}_{w2e}$, $\mathbf{Q}_{e2w}$ and $\mathbf{Q}_{e2e}$ are query matrices, and $\mathbf{K}$ is the key matrix.

It is beneficial to use the information of entity tokens when computing the attention scores. However, at the same time, we would like to avoid the quadratic attention complexity attached to long text sequences (Figure 3.3(a)). In the LUKE implementation, each kind of attention score between pairs of tokens is computed separately because



(a) LUKE attention (b) Long-LUKE attention

Figure 3.3: Proposed configuration of Attention patterns. Color code of attention pattern: Orange - $word2word$, Green - $word2entity$, Blue - $entity2word$, Purple - $entity2entity$

each has its query and key matrices. Only then are the scores concatenated, fed into a Softmax, and multiplied by the Value matrix to obtain the final attention score. With this in mind, we acknowledge that the attention scores associated with entities have at most $\mathcal{O}(n)$ complexity for a sequence of n tokens. It would be beneficial to replace the word-to-word attention pattern, which causes $\mathcal{O}(n^2)$ complexity. This implementation does not modify the attention pattern of the entity-related scores.

Consequently, we used a non-dilated sliding window attention pattern from a Long-Transformer (Figure 3.3(b) - orange), reducing the complexity to $\mathcal{O}(n \times w)$. In the context of our combined attention pattern (Figure 3.3(b)), we considered the entity-related attention patterns as "global attention" in the sense that the attention operation is symmetric, meaning a fixed entity token attends to all tokens across the sequence, and all tokens in the sequence attend to it. This global attention is more dynamic than the one from Long-Transformer, meaning a broader range of tokens appears several times in the text. We hypothesize that such global attention improves the ability of our model to deal with entity-related tasks. Figure 3.3(b) displays an example of our combined entity-aware sliding window attention with "global attention" on entity tokens. Note that the number of $w2w$ tokens in our implementation is larger than the counterparts. Therefore we found that only this operation was required to reduce its complexity. The attention operation that we used, inspired by Longformer, has a *chunking* operation that requires that matrix result of the matmul of the query and key matrices to contain the proprieties of a square matrix multiple of 512. This condition is only met when the *matmul* operation is done between Q_{w2w} and K_{w2w} matrices (i.e., attention between words). In order to apply this operation to the few remaining cases, we would have to either: (1) apply padding to all query and key matrices for every case, which would

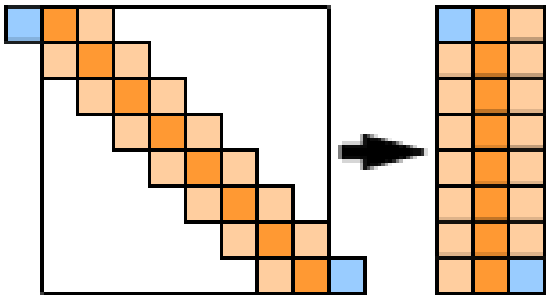


Figure 3.4: Representation of the chunking operation applied to Attention between words.

introduce a large number of dummy values, as well as increase the overall computation and memory complexity; or (2) completely redesign the format of the matrices and the attention operations. Instead, we applied the *chunking* (Figure 3.4) operation only to the suitable case and later applied a conversion technique so that all attention matrices are compatible to be concatenated. The idea behind the *chunking* operation is to compute the scores only in the positions covered by the sliding window, completely discarding the remaining computations (blank area on Figure 3.4). For an example of a sequence length of 1024 tokens, LUKE would compute the attention for each token with every other sequence token. In Long-LUKE, the idea is to compute the attention between only the relevant pairs of tokens and reduce the correlated complexity. Consequently, we concatenate the attention scores for all cases to obtain the overall attention_scores that will be normalized, fed through a *Softmax* and multiplied by the Value layer (**V**), resulting in the output of the self-attention operation.

The main advantage of Long-LUKE, as displayed in Figure 3.1, is the ability to maintain the input representation and the architecture and enable it now to receive longer sequences of text in the same tasks. Using the entity representations from Long-LUKE and the custom self-attention implementation, we built different approaches to address the Entity typing and Relation classification tasks.

3.3 Overview

Section 3.1 of this Chapter expands upon the structural knowledge required to understand the Long-LUKE self-attention, allowing Section 3.2 to focus on detailing the implementation we developed. A particular emphasis was placed on adapting LUKE's self-attention through a custom implementation. We specify the steps, the complications we faced during the implementation, and the appropriate solution we found.

Chapter 4

Evaluation

In Section 4.1, we present the datasets we used for testing, some baseline models, and the evaluation metrics we used to evaluate the performance. Next, we present the experimental setup in Section 4.2, as well as the results of our model in the Entity Typing (Section 4.3) and Relation Classification (Section 4.4) tasks.

4.1 Baselines, Datasets and Evaluation Methodology

This Section introduces the baseline models used to compare the performance of our implementation alongside listing the adequate datasets to execute the chosen experiments. Finally, we will detail the specific metrics used to evaluate each task.

4.1.1 Baselines

In order to understand whether there is an actual improvement in our implementation, we have to compare our developed model with the ones it is based on, specifically LUKE (Yamada et al., 2020) and RoBERTa (Liu et al., 2019).

Infusing Knowledge into Pre-Trained Models with Adapters (K-Adapter), proposed by Wang et al. (2021) is another baseline we will use. It addresses the limitation there is when we insert multiple sources of knowledge into a PLM like BERT or RoBERTa (i.e., models hurdle to capture rich factual knowledge (Kirkpatrick et al., 2017)). This is achieved with the introduction of Adapters, knowledge-specific models annexed outside of a PLM that receive as input the output of hidden states of intermediate layers of the PLM. As adapters are not a module of the PLM and have few parameters, their training is memory efficient. K-Adapter is tested on ET and RC to explore the ability of models to learn factual knowledge.

On the other hand, to evaluate the impact of the techniques that aim to reduce the time and memory complexity of the self-attention mechanism, we will also use the Document-level Relation Extraction as Semantic Segmentation (DocuNET) (Zhang et al., 2021) model as a baseline. They present current state-of-the-art results in NLP tasks related to long documents.

4.1.2 Datasets and Evaluation Metrics

In This Subsection we detail the short and long-sequenced datasets, and explain why we chose which dataset for each task. Alongside this, we present statistics that give us some context. Finally, we explicit what metrics we used for each dataset.

4.1.2.1 Short-Sequenced Baseline Datasets

As general evaluation datasets, we chose OpenEntity (Choi et al., 2018) and FIGER (Ling and Weld, 2012) to test our model in the ET task, and TA-CRED (Zhang et al., 2017) in the RC task. OpenEntity includes 6,000 sentences sampled from web articles. In contrast, FIGER is more fine-grained, covering 113 unique ET tags. This dataset was developed by exploiting the anchor links in Wikipedia text to automatically label entity segments with appropriate tags. TA-CRED, on the other hand, contains 119,474 sentences, covering 42 relation types between entities. These datasets are not very good baselines to evaluate the performance of a model over large sequences of text due to their low average number of words per sample. Nevertheless, they serve as a baseline to verify whether Long-LUKE behaves the same as LUKE.

4.1.2.2 Long-Sequenced Baseline Datasets

Most existing datasets focus on identifying relationships between more extended corpus. Using a large-scale dataset at the document level is relevant since it requires an understanding of the whole document to annotate entities and their document-level relationships that span beyond sentences. As A Large-Scale Document-Level Relation Extraction Dataset (DocRED) (Yao et al., 2019) and Revisiting DocRED - Addressing the Overlooked False Negative Problem in Relation Extraction (ReDocRED) (Tan et al., 2022) satisfied this criterion, we used these datasets to evaluate the ET and RC tasks on Long-sequenced documents. Both datasets are constructed from Wikipedia and Wikidata and annotate both named entities and relations. They require reading multiple sentences in a document to extract entities and predict their labels by synthesizing

all information from the document. We find that both DocRED and ReDocRED are more extensive than existing datasets in many aspects, including the number of documents, words, sentences, and entities, especially in aspects of relation types, relation instances and relational facts (see Table 4.1 and 4.2). DocRED includes 96 common relation types from Wikidata, covering a wide range of categories: science, art, time, personal life, and more. Concerning the data split, DocRED contains 3,053 training examples, 998 validation examples and 1,000 non-labelled evaluation examples.

An important detail in DocRED is that an entity usually contains more than one mention occupying different positions in the document. For example, if in a document we have the mentions “Portugal” and “República Portuguesa”, they represent the same entity “Portugal”, even though they are two different words in the document. In contrast with other RE datasets such as TA-CRED, DocRED has entity linking, meaning all mentions are grouped. Therefore, a relation between 2 pairs of entities in DocRED can be seen by LUKE as a relation between several pairs of mentions (i.e., in DocRED, a relation between an entity A with m mentions and an entity B with n mentions result in $m \times n$ pairs of relations between mentions to be fed into LUKE). The evaluation was performed through the https://codalab.lisn.upsaclay.fr/competitions/365 platform.

Consequently, we had to extend the training dataset used in the finetune of our model for this task, following Zhang et al. (2021) evaluation. The labels in the training dataset only contained examples where there is a relation, disregarding the cases where there is no relation. To include the missing negative examples, we considered all the relations between every mention of each example and marked their relation as N/A whenever they were missing from the original dataset. Consequently, the number of negative labels is very high, as seen in Table 4.1. We now have 1,198,650 training examples and 392,158 evaluation examples. Due to not having entity relation labels in the test dataset, we need to train DocRED using both the 96 positive and the negative labels and predict every positive relation between every entity in each document, as expected by the official CODALAB evaluation. Due to the many NA cases in the dataset, if we finetune our model with an unbalanced dataset split, all predictions can be perceived as NA, damaging the model’s performance.

We can note that DOCRED is challenging for existing RE methods, which signifies that document-level RE remains an open problem that requires further efforts. Tan et al. (2022) found that DocRED is incomplete for relation extraction tasks due to the several

false negative samples, therefore developed ReDocRED (Tan et al., 2022), which was built by re-annotating over four thousand documents in DocRED dataset by adding the missed relation triples. For the relation extraction task, ReDocRED introduced an additional classification label (N/A) in the evaluation dataset, meaning the test set now contains all positive and negative relation labels, which does not happen in the CODALAB official evaluation task for the DocRED dataset. Specifically, ReDocRED built its evaluation dataset from 500 of the 1000 documents present in the *dev* dataset of DocRED. By doing this, all labels are now accessible. From Table 4.1, ReDocRED introduced for the same documents over 70,000 new triples.

4.1.2.3 Evaluation Metrics

To evaluate the tasks to be performed, we will take into account different evaluation metrics, namely:

1. Accuracy (Acc) is the quotient between correct predictions and the total number of predictions.

2. Precision (P) measures the accuracy of the predictions delivered, dividing the number of correctly predicted observations (i.e., for ET the ground truth and predicted types for a target entity match) with the total number of observations.

3. Recall (R) stands for the ratio of correctly predicted positive observations to all observations in the actual class. We use this metric to select the best model when a high cost is associated with False Negatives.

4. F_1-score (F_1) is the weighted average of P and R.

Dataset	# Docs	# Labels	N/A	# Triples	# N/A Training	Avg. # Entities	Avg. #Pairs per Document
DocRED	3053	96*	97.1%	50 503	840 000	19.5	393.6
ReDocRED	3053	97	94.0%	120 664	224 146	19.4	391.0
TA-CRED	119 474	42	79.9%	21 373	94 026	2	1

* The model is trained using 97 labels to predict the triples, but the official CODALAB evaluation only contains 96 labels.

Table 4.1: Statistics of Relation Classification Datasets. We can see that the re-annotation efforts on sentence-level relation extraction datasets (DocRED $\rightarrow$ ReDocRED) result in smaller percentages of no_relation (N/A) samples, indicating that false negative is a common problem in the RC dataset.

Dataset	# Docs	# Labels	Avg. # Entities	Avg. # Sents	Avg. # Words	Max. # Words
OpenEntity	6 000	9	2.3	1	29.1	120
FIGER	2 010 563	113	4.4	1	29.3	52
DocRED	3053	7	19.5	8.1	223.1	563

Table 4.2: Statistics of Entity Typing Datasets.

The computation of the metrics for binary classification is based on a confusion matrix (Table 4.3) and is defined according to the following equations:

$$\text{Acc} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}, \tag{4.1}$$

$$\begin{aligned} \text{P} &= \frac{\text{TP}}{\text{TP} + \text{FP}}, \\ \text{R} &= \frac{\text{TP}}{\text{TP} + \text{FN}}, \\ F_1 &= 2 \times \frac{\text{P} \times \text{R}}{\text{P} + \text{R}}. \end{aligned} \tag{4.2}$$

For evaluating multi-class scenarios, we adopt averaging methods for the F_1 score calculation, resulting in different average scores (i.e., macro, weighted, and micro). In order to evaluate our tasks that contain multiple classes, instead of having multiple per-class F_1 scores, it is preferable to average every score and obtain a single number to describe the overall performance. In order to accomplish this, we can apply three different types of average F_1 score:

- The macro-averaged F_1 score is defined as the arithmetic mean of all the per-class F_1 scores, where all classes are treated equally independently of their support values;

- The weighted-average F_1 score is obtained by taking the mean of all per-class F_1 scores, taking into account the contribution of each class as weighted by the number of examples of that given class.

- The Micro averaging computes a global average of F_1 score by counting the sums of the TP, FN, and FP (Equation 4.2). Essentially, this computes the proportion of correctly classified observations out of all observations.

Our main objective is to excel in LUKE's performance achieved in the ET and RC tasks.

		True Label	
		Type of target entity	Non Type of target entity
Predicted label	Type of target entity	True Positive (TP)	False Positive (FP)
	Non Type of target entity	False Negative (FN)	True Negative (TN)

Table 4.3: Confusion Matrix for the Entity Typing task.

4.2 Experimental Setup

This Section starts by introducing the architecture of Long-LUKE and what was done to convert LUKE into a Long-Transformer. What succeeds is a listing of the values of the hyperparameters alongside some relevant training details.

4.2.1 Model Architecture

Long-LUKE is based on BERT (Devlin et al., 2019a), containing $D = 1024$ hidden dimensions, 24 hidden layers, 16 self-attention heads each with 64 head dimension, and 256 entity token embeddings. The input text is tokenized into words using RoBERTa's tokenizer with the vocabulary consisting of Vw = 50K words and the Ve = 500K most frequently appearing entities in the entity annotations. The parameters on our word and entity embeddings are $483M$ and $128M$, respectively.

The implementation is based on Huggingface and Pytorch. We used the finetuned models of LUKE-Large as the base for our work and adapted it into a Long-Transformer without additional training. We extend the limit sequence limit to 1024 tokens (i.e., initializing the additional position embeddings by copying the embeddings of the first positions) and alter the implementation of the self-attention operations within each layer while keeping the model parameters. In theory, by doing this, we should be able to model entities in extended sequence examples, which is especially important to perform relation extraction between distant entity pairs. Our model is later trained to predict relations between pairs of entities in the same or different sentence or to predict the type of entity according to the context surrounding the target entity.

Specifically, we started from the finetuned *Huggingface* Transformer models of *studio-ousia/luke-large* and adapted it to a Long-Luke through the procedure described in Section 3.2.2. From the Longformer implementation, we chose to use the regular sliding window attention pattern instead of the dilated sliding window pattern as it was only used in a PyTorch memory-efficient slow implementation and another implementation using an optimized custom CUDA kernel implemented using TVM (Chen et al., 2018), targeted for the language modelling experiments. Due to time and memory restraints, we could not explore the TVM implementation. The Pytorch-friendly implementation, used for the pre-training/finetuning setting, only supports the non-dilated case. We also decided to apply the global attention strategy only to entity-related tokens.

4.2.2 Hyperparameters Tuning

We tuned the hyperparameters over the development dataset, performed all experiments on one Quadro RTX 8000 graphics card, and evaluated our model following the metrics of each dataset documentation. Regarding the training of our model for each dataset and experiment, we tried to use the hyperparameters cited in Vaswani et al. (2017). We chose sliding window size $= 256$ for all layers, maximum word length $= 1024$, batch size $= 64$, learning rate $= 1e-5$, and the *Adam* optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e-6$. We used *Adafactor* instead of *Adam* because it reduces memory usage, maintaining a similar behaviour to *Adam*.

4.2.3 Training Setup for the Tasks

For the ReDocRED relation classification task, we finetuned Long-LUKE with *Adafactor* using learning rates $5e-5$ over 7 epochs and evaluated our model with standard F_1, following Yao et al., 2019. On the remaining datasets, we finetuned our model over 3 epochs. For every experiment, we finetuned the model with a multi-class Linear classification head on top, having the input features be the output of Long-LUKE, and the output features the number of classes of each dataset. Additionally, for the ReDocRED RC task, to obtain the relation between entities, we choose the relation between the entity pair following the more representative relation between mentions. The reason we do this is that we need to have an entity span assigned to the entity to be fed into the model, and such representation can only be found in the mentions of each entity, it is not linked to the entity itself initially.

4.3 Entity Typing Task

We performed our first experiments on entity typing, which is the task of identifying the type of an entity given a sentence. We use the Open Entity (Choi et al., 2018), the FIGER (Ling and Weld, 2012), and the DocRED (Yao et al., 2019) datasets. We did not perform the ET task on the ReDocRED dataset because it shares the same Entity Types labels with the DocRED dataset. We report loose micro-precision, recall, and F1 for both datasets and employ the micro-F1 as the primary metric. The statistics of the entity typing task can be seen in Table 4.2.

4.3.1 Results on the FIGER Dataset

We evaluated LUKE and Long-LUKE on this task and compared the results with other transformer-based models, including K-Adapter (Wang et al., 2021), and RoBERTa (Liu et al., 2019). From Table 4.4, we observe that both LUKE and Long-LUKE outperformed the baselines by 8 micro-F_1 points. This performance increase is mainly due to the impact feeding entities into the model has. Our model performs the same as LUKE, which is expected since, in this dataset, no example surpasses the 512 tokens limit. This validates that Long-LUKE behaves the same as the original LUKE for the defined window size whenever the maximum sequence of examples does not surpass the original maximum length.

	FIGER (entity typing)		
Method	Acc	macro-F_1	micro-F_1
LUKE	**64.73**	**88.28**	**88.26**
Long-LUKE	**64.73**	**88.28**	**88.26**
K-Adapter	59.5	84.52	80.42
RoBERTa	56.31	82.43	77.83

Table 4.4: Evaluation metrics for the Entity Typing FIGER task

4.3.2 Results on the OpenEntity Dataset

We evaluate an ET task for this dataset based on the corresponding entity representation. From Table 4.5, both Long-LUKE and LUKE outperform the transformer-based baseline models by 0.6 micro-F_1, including K-Adapter (Wang et al., 2021) and RoBERTa (Liu et al., 2019). The improvement of the Precision and Recall values displays that modelling entities contribute to the model to return more relevant results than irrelevant ones. Also, the increased recall value symbolizes that the model returns more relevant results. Both LUKE and Long-LUKE achieved the same performance values for this dataset, and this is because OpenEntity does not have examples whose sequences are longer than 512 tokens. The document with the most extended sequence has 120 words. In this context, Long-LUKE also replicates LUKE.

	OpenEntity (entity typing)		
Method	P	R	micro-F_1
LUKE	**79.69**	**76.5**	**78.1**
Long-LUKE	**79.69**	**76.5**	**78.1**
K-Adapter	79.3	75.8	77.5
RoBERTa	77.6	75.0	76.2

Table 4.5: Evaluation metrics for the Entity Typing OpenEntity task

4.3.3 Results on the DocRED Dataset

About 150 documents of DocRED have text sequences longer than the usual 512 tokens most models can deal with, and the average number of tokens per document is 305. Consequently, this dataset is ideal for evaluating the performance of our implemented Long-Transformer self-attention suitable for longer sequences on tasks such as relation classification. However, in my implementation, I used an unbalanced training dataset for the finetuning of the RC tasks, containing too many N/A examples (840k N/A, Vs 50k positives), which caused all relations to be predicted as being N/A in the official evaluation. Due to not being able to filter the positive relations, the unavailability of the real relation test labels, the large number of training examples, as well as our GPU memory and time limitations, we found that it would be too costly to retrain both our model and our baselines on this dataset. As an alternative, we decided to evaluate DocRED on a different task other than relation classification. We evaluated Long-Document Entity Typing, taking advantage of the dataset's available labels for the types of entities. From Table 4.6, both LUKE and Long-LUKE obtain up-to-standard results. Comparing both implementations, Long-LUKE has access to information that LUKE does not. Long-LUKE can access a greater range of context, enabling the model to identify the target entity type successfully. Such differences are noticeable in 1.81 points increase in Precision, Recall, and F_1. Despite Long-LUKE improving over LUKE, we do not have other baselines to compare with.

	DocRED (entity typing)		
Method	P	R	F_1
Long-LUKE	**73.09**	**73.09**	**73.09**
LUKE	71.28	71.28	71.28

Table 4.6: Evaluation metrics for the Entity Typing DocRED task

4.4 Relation Classification Task

We performed the following experiments on relation classification, which is the task that assigns the correct relation between *head* and *tail* entities in a document. We used the TA-CRED dataset (Zhang et al., 2017), a large-scale intra-sentence relation classification dataset, as well as a dataset focused on inter-sentence relations, named ReDocRED (Tan et al., 2022). The statistics of these datasets are in Table 4.1.

4.4.1 Results on the TA-CRED Dataset

The experimental results for this dataset are in Table 4.7. Both Long-LUKE and LUKE outperform our established baselines by 1.4 micro-F_1 points. TA-CRED only contains concise sequenced examples of a maximum of 50 tokens, which does not enable us to evaluate the performance over longer sequences accurately. Nevertheless, we verified that Long-LUKE performs the same as the original LUKE, and that the introduction of entities was crucial in improving the Precision, Recall, and F_1 scores.

	TA-CRED (relation extraction)		
Method	P	R	micro-F_1
LUKE	70.4	75.1	**72.7**
Long-LUKE	70.4	75.1	**72.7**
RoBERTa	70.2	72.4	71.3
BERT-base	67.2	64.8	66.0

Table 4.7: Evaluation metrics for the TA-CRED Dataset

4.4.2 Results on the ReDocRED Dataset

We obtained the experimental results using 206,933 negative samples and about 120,000 positive samples due to the extension of the dataset. We chose to apply this data split to all baselines in this dataset to balance our training. Concerning the results in Table 4.8, it is noticeable that both LUKE and Long-LUKE benefit from having entity representations since they delivered higher F_1 scores compared to their baseline counterparts. Also, we considered every relation between entities three sentences apart from each other as having the N/A relation. This technique helps us discard the relations between pairs of entities with a low probability of being related.

Furthermore, Long-LUKE obtained 2.38 more F_1 points, compared to LUKE, due to its ability to cover more expansive sequences of texts and acquire a richer understanding of the context of the entity pair. Note that there are models with a higher F_1

score for this task, but their architectures are very different. As DocuNET (Zhang et al., 2021) is based on the RoBERTa-large implementation, which is the same as LUKE and Long-LUKE, we found this baseline to be the most suitable one.

	ReDocRED (relation extraction)
Method	F_1
LUKE	82.64
Long-LUKE	**85.02**
BERT-base	53.20
DocuNET-RoBERTa-large	79.09

Table 4.8: Evaluation metrics for the ReDocRED Dataset

4.5 Overview

In this Chapter, we presented the baseline models, and the datasets, alongside their evaluation methodology. Next, we displayed the training setting and the results of Long-LUKE for the ET and RC tasks, showing a competitive performance against state-of-the-art entity-oriented models.

The results hinted that even though we discarded very distant pairs of entities in the relation classification task, we still obtained better scores than the LUKE model. Such indicates that Long-LUKE makes good use of the additional context it acquires when covering longer text sequences, even when entities are adjacent. Having two entities close to each other is not enough condition to obtain an accurate prediction. The context, vital to finding the solution in these tasks, may be located in a section of the text unavailable beforehand due to the short input sequence limit. Furthermore, other NLP tasks can benefit from covering more text in a document to acquire richer context, which is challenging when we model shorter document spans.

Chapter 5

Conclusions and Future Work

This Chapter presents the closing arguments of the dissertation, as well as an overall view of the accomplished work, stating the main conclusions that can be drawn from the experimental results obtained and how the work this dissertation entails can be expanded upon in the future.

5.1 Summary of Contributions

Throughout this thesis, we developed our work around central objectives. One of them is to verify if Transformer (Vaswani et al., 2017) models improved their results whenever they consider entities as input. Also, we proposed to check if changing the self-attention pattern benefits the model by increasing its performance on multiple essential NLP tasks that involve entities. On top of that, we wished to overcome one of LUKE's model limitations, the inability to deal with long-sequenced documents, mainly due to the quadratic complexity involved in the self-attention operations. Having this in mind, we proposed a new entity-aware method for Entity Typing and Relation Classification, extending the previous LUKE (Yamada et al., 2020) and Long-Transformer (Beltagy et al., 2020) approaches. We obtained results that are on par with state-of-the-art supervised methods over multiple challenging datasets while also generalizing across different domains and lengths of Documents.

Another contribution was the confirmation that it is reliable to convert a finetuned model and convert it into a Long-Transformer, without taking the risk of suffering a decrease in performance. Such confirms that it is possible to enhance the existing finetuned models by assigning them new perks and abilities without re-training them. Furthermore, on a research level, the overall theme of the thesis should motivate to

apply the same principle to develop implementations which tackle real-world limitations in the same way Long-LUKE did.

Overall, by testing our model over multiple exciting datasets and achieving favourable results, this thesis has shown that having the representation of entities and the richer context surrounding such entities significantly benefit models that look for solving tasks involving entities and their complex relations. We can take advantage of the context surrounding the target to predict cases that are not trivial.

5.2 Future Work

For future work, we can consider other methods for handling long inputs besides the Longformer (e.g., memory efficient attention implementations (Rabe and Staats, 2021), or other sparse attention patterns such as those in the Hypercube Transformer (Wang et al., 2022) or Big Bird (Zaheer et al., 2020)).

It would also be interesting to apply the pre-training task of ERICA (Qin et al., 2021a) to our model. This task would improve the model's knowledge of inter-sentence entity relations, and it should be able to increase the evaluation scores on the relation extraction task of ReDocRED. Also, note that our model was not pre-trained over more data. It would be beneficial to run a task similar to MLM with extended sequenced examples. By doing this, the model would further its knowledge of entity saliency and entity relations, which we believe would increase the scores of all tasks overall. Also, in further experiments, we would evaluate our model on the OntoNotes 5.0 (Weischedel et al., 2022), LongtoNotes (Shridhar et al., 2022), and Long-Range Arena (Tay et al., 2021) datasets to test the performance limit of our model, as the datasets contain several examples which sequences surpass 1024 tokens, the number we defined as the most extensive sequence to be fed into our model. The idea would be to stress further the length of the textual inputs that need to be analyzed.

Another idea for future work is to evaluate our model on other relevant tasks, such as keyphrase extraction or Text Anonymization

. Finally, one could also evaluate the performance of entity-related models on the ReDocRED dataset without the negative examples, meaning all the samples of targeted pairs of entities would have an existing relation. Despite being an interesting experiment, we could not execute it due to time constraints.

Bibliography

Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). Layer Normalization. *Computing Research Repository (CoRR) abs/1607.06450*.

Baldini Soares, L., N. FitzGerald, J. Ling, and T. Kwiatkowski (2019). Matching the Blanks: Distributional Similarity for Relation Learning. In *Proceedings of the Association for Computational Linguistics (ACL)*, pp. 2895–2905.

Beltagy, I., M. E. Peters, and A. Cohan (2020). Longformer: The Long-Document Transformer. *Computing Research Repository (CoRR) abs/2004.05150*.

Blei, D. M., A. Y. Ng, and M. I. Jordan (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research Volume 3*.

Chen, T., T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy (2018). TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, pp. 579–594.

Choi, E., O. Levy, Y. Choi, and L. Zettlemoyer (2018). Ultra-Fine Entity Typing. In *Proceedings of the Meeting of the Association for Computational Linguistics (MACL) (Volume 1: Long Papers)*, pp. 87–96.

Clark, K., M. Luong, Q. V. Le, and C. D. Manning (2020). ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. *Computing Research Repository (CoRR) abs/2003.10555*.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2019a). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL): Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2019b). Multilingual BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the Conference of the Association for Computational Linguistics (ACL): Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186.

Ettinger, A. (2020). What BERT Is Not: Lessons from a New Suite of Psycholinguistic Diagnostics for Language Models. *Transactions of the Association for Computational Linguistics (ACL) volume 8*, pages 34–48.

He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

He, P., X. Liu, J. Gao, and W. Chen (2021). DeBERTa: Decoding Enhanced BERT With Disentangled Attention. In *Computing Research Repository (CoRR)*, Volume abs/2006.03654.

Jiang, Z.-H., W. Yu, D. Zhou, Y. Chen, J. Feng, and S. Yan (2020). ConvBERT: Improving BERT with Span-based Dynamic Convolution. *Advances in Neural Information Processing Systems volume 33*, pages 12837–12848.

Joko, H., F. Hasibi, K. Balog, and A. P. de Vries (2021). Conversational Entity Linking: Problem Definition and Datasets. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2390–2397.

Joshi, M., D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy (2020). SpanBERT: Improving Pre-training by Representing and Predicting Spans. *Transactions of the Association for Computational Linguistics (TACL)*, 64–77.

Kingma, D. P. and J. Ba (2015). Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.

Kirkpatrick, J., R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell (2017). Overcoming Catastrophic Forgetting in Neural Networks. *In Proceedings of the National Academy of Sciences volume 114*, pages 3521–3526.

Kitaev, N., L. Kaiser, and A. Levskaya (2020). Reformer: The Efficient Transformer. *Computing Research Repository (CoRR) abs/2001.04451*.

Lan, Z., M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut (2020). Albert: A Lite BERT for Self-supervised Learning of Language Representations. In *International Conference on Learning Representations*.

Ling, X. and D. S. Weld (2012). Fine-Grained Entity Recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 94–100.

Liu, Y., M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *Computing Research Repository (CoRR) volume 364*.

McCloskey, M. and N. J. Cohen (1989). Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. Volume 24 of *Psychology of Learning and Motivation*, pp. 109–165.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks Journal volume 12*, pages 145–151.

Qin, Y., Y. Lin, R. Takanobu, Z. Liu, P. Li, H. Ji, M. Huang, M. Sun, and J. Zhou (2021a). ERICA: Improving Entity and Relation Understanding for Pre-trained Language Models via Contrastive Learning. In *Proceedings of the Meeting of the Association for Computational Linguistics (MACL) and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 3350–3363.

Qin, Y., Y. Lin, R. Takanobu, Z. Liu, P. Li, H. Ji, M. Huang, M. Sun, and J. Zhou (2021b). ERICA: Improving Entity and Relation Understanding for Pre-trained Language Models via Contrastive Learning. In *Proceedings of the Meeting of the Association for Computational Linguistics (MACL) and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 3350–3363.

Rabe, M. N. and C. Staats (2021). Self-attention Does Not Need O(n^2) Memory. *Computing Research Repository (CoRR) abs/2112.05682*.

Ri, R., I. Yamada, and Y. Tsuruoka (2021). mLUKE: The Power of Entity Representations in Multilingual Pretrained Language Models. *Computing Research Repository (CoRR) abs/2110.08151*.

Ruder, S. (2017). An overview of Gradient Descent optimization algorithms. *Computing Reasearch Repository (CoRR)*.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). Learning Representations by Back-propagating Errors. *Nature Journal volume 323*, 533–536.

Shridhar, K., N. Monath, R. Thirukovalluru, A. Stolfo, M. Zaheer, A. McCallum, and M. Sachan (2022). Longtonotes: Ontonotes with Longer Coreference Chains. *Computing Research Repository (CoRR) abs/2210.03650*.

Tan, Q., L. Xu, L. Bing, and H. T. Ng (2022). Revisiting DocRED - Addressing the Overlooked False Negative Problem in Relation Extraction. *Computing Research Repository (CoRR)*.

Tay, Y., M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler (2021). Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*.

Tieleman, T., G. Hinton, et al. (2012). Lecture 6.5 - RmsProp: Divide The Gradient By a Running Average Of Its Recent Magnitude. *COURSERA: Neural Networks for Machine Learning volume 4*, pages 26–31.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). Attention is All You Need. In *Proceedings of the International Conference on Neural Information Processing Systems*, pp. 6000–6010.

Wang, R., D. Tang, N. Duan, Z. Wei, X. Huang, J. Ji, G. Cao, D. Jiang, and M. Zhou (2021). K-Adapter: Infusing Knowledge into Pre-Trained Models with Adapters. In *In Proceedings of the Association for Computational Linguistics (ACL)*, pp. 1405–1418.

Wang, Y., C.-T. Lee, Q. Guo, Z. Yin, Y. Zhou, X. Huang, and X. Qiu (2022). What Dense Graph Do You Need for Self-Attention? In *Proceedings of the 39th International Conference on Machine Learning*, Volume 162, pp. 22752–22768.

Weischedel, R., M. Palmer, M. Marcus, H. Eduard, S. Pradhan, L. Ramshaw, N. Xue, A. Taylor, J. Kaufman, M. Franchini, M. El-Bachouti, R. Belvin, and A. Houston (2022). OntoNotes Release 5.0 - LDC2013T19.

Yamada, I., A. Asai, H. Shindo, H. Takeda, and Y. Matsumoto (2020). LUKE: Deep Contextualized Entity Representations with Entity-aware Self-attention. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6442–6454.

Yao, Y., D. Ye, P. Li, X. Han, Y. Lin, Z. Liu, Z. Liu, L. Huang, J. Zhou, and M. Sun (2019). DocRED: A large-scale Document-Level Relation Extraction dataset. In *Proceedings of the Meeting of the Association for Computational Linguistics (MACL)*, pp. 764–777.

Zaheer, M., G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed (2020). Big Bird: Transformers for Longer Sequences. In *Advances in Neural Information Processing Systems*, Volume 33, pp. 17283–17297.

Zhang, N., X. Chen, X. Xie, S. Deng, C. Tan, M. Chen, F. Huang, L. Si, and H. Chen (2021). Document-level Relation Extraction as Semantic Segmentation. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*.

Zhang, W., Y. Feng, F. Meng, D. You, and Q. Liu (2019). Bridging the Gap between Training and Inference for Neural Machine Translation. In *Proceedings of the Meeting of the Association for Computational Linguistics (MACL)*, pp. 4334–4343.

Zhang, Y., V. Zhong, D. Chen, G. Angeli, and C. D. Manning (2017). Position-aware Attention and Supervised Data Improve Slot Filling. In *Proceedings of Empirical Methods in Natural Language Processing (EMNLP)*, pp. 35–45.