Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik 6
Prof. Dr.-Ing. Hermann Ney

Selected Topics in Human Language Technology and Machine Learning im WS2023

Unsupervised Training of Language Representations

Ahmed Ali

Matriculation Number 414360

March 22, 2023

Supervisor: David Thulke

Contents

1 Introduction

Natural Language Processing (NLP) has seen a surge in popularity due to the use of unsupervised training of language representations. This technique does not require explicit supervision or annotated data, making it ideal to leverage large amounts of unannotated data which is more abundant than annotated data. Recent language representation models, such as autoencoders, word2vec, BERT, and GPT, use unsupervised learning approaches to learn language representations. This involves predicting the context of a word or sentence given its surrounding context. Unsupervised training of language representations has many applications in Natural Language Processing (NLP) such as text classification, language translation, sentiment analysis, and information retrieval. This technique has demonstrated the capacity to perform better than supervised methods in certain cases, and its ability to learn from data that is not labeled makes it highly flexible and applicable to various domains.

The use of language models in everyday life has prompted us to consider the potential applications of such models. Language models are employed in a variety of activities, such as providing word predictions during digital conversations and offering related paragraphs in response to search engine queries through Question Answering. This raises the question of what other possibilities can be explored with models that possess a deep understanding of language.

Unsupervised training of language representations has proven to be an efficient way to address data sparsity due to its scalability and applicability. This approach can be especially helpful in the case of limited or expensive annotated data for low-resource languages or specialized domains, as it allows for leveraging large amounts of unannotated data to learn meaningful language representations, which can then be used to enhance downstream NLP tasks.

Transfer learning is an important concept in unsupervised training of language representations. Pre-training a language model on a large amount of unannotated data allows the learned representations to be applied to other NLP tasks with minimal fine-tuning or additional annotated data. Such advancements have led to noteworthy improvements in performance and enabled the development of powerful language models, like GPT-3. However, unsupervised training of language representations is not without its challenges. The quality of the learned representations is highly dependent on the quality and diversity of the training data. Additionally, the training process can be computationally expensive, requiring large amounts of computing power and time. Finally, the interpretability of the learned representations is an ongoing challenge, as it can be difficult to understand how the model is representing language and what factors are contributing to its performance.

It is crucial to differentiate between language models (LMs) and language representations in the field of natural language processing (NLP). While LMs predict the probability of a sequence of words, given a sentence, language representations refer to models that capture the meaning and context of the words within a sentence. In this paper, we will focus specifically on the training and fine-tuning of language representations for tasks such as question-answering. We will also discuss the training and fine-tuning of non-English language models, specifically focusing on the Arabic language. The challenges associated with applying these processes to a language other than English will be examined.

2 Preliminaries

2.1 Language Models history

This section presents an overview of the major language modeling techniques developed over the past decade, with a focus on how each technique has advanced the field. While not comprehensive, the aim is to provide the reader with a basic understanding of each technique and how it has built on prior developments. A concise description of each approach will be included to assist in forming a better understanding of the primary focus of this work.

The Bag of Words (BoW) technique is useful for converting words into numerical representations by analyzing their presence in a given sentence. The encoded value of a word corresponds to the number of times it appears in the sentence; for binary BoW, a word is denoted as 1 if present and 0 otherwise. This technique is beneficial for topic modeling, yet fails to capture the relationship between words, rendering it inadequate for tasks such as sentiment analysis.

ELMo, or Embeddings from Language Models, is a bi-directional Long Short-Term Memory (LSTM) [22] language model that can generate deep representations that take into account all the internal layers of the bidirectional architecture [18]. The authors note that ELMo can be easily integrated into existing models and fine-tuned for various tasks such as question-answering, sentiment analysis, and textual entailment. However, due to its use of LSTMs, ELMo has difficulty capturing dependencies for longer phrases.

OpenAI then presented the GPT model as a decoder-only model, with GPT-3 being unidirectional. While GPT-3 differs from ELMo in architecture, it has certain limitations. For instance, when synthesizing text, GPT-3 samples may repeat themselves semantically at the document level, have incoherence over long passages, or contain non-sequitur sentences or paragraphs [7].

In recent years, various language models have been introduced, such as BERT, T5, and GPTchat [20]. We will evaluate BERT's performance in downstream tasks and compare it to other models.

2.2 Sequence-To-Sequence models

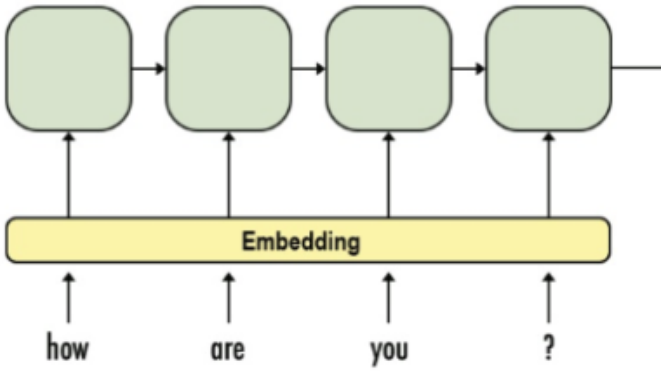


Figure 1: Sequence-to-Sequence encoder

Seq2Seq [23] models are a type of neural network that is designed to transform input

sequences into output sequences of varying lengths. Initially created for machine translation, they have since been utilized in various tasks, including speech recognition, text summarization, and conversational AI.

Utilizing a pair of recurrent neural networks (RNNs), namely an encoder Figure 1 and a decoder, is a common approach for Seq2Seq models. This enables the conversion of input sequences to output sequences. The encoder RNN reads the input sequence and produces a fixed-length vector representation of the input, which is then passed to the decoder RNN. The decoder RNN generates the output sequence one element at a time, utilizing the fixed-length vector representation from the encoder RNN as its initial state. Seq2Seq models are advantageous for tasks such as machine translation due to their capacity to accommodate input and output sequences of varying lengths, allowing for a seamless transition between source and target languages.

At time step i, the output of the hidden state in the encoder is represented by the last hidden state as follows:

$$h_i = f(W^{(hh)}h_{i-1} + W^{(hx)}x_i)$$

where x_i is the ith token and W^{hh} is the weight matrix between the two hidden states h_i and h_{i-1}. W^{hx} is the weight matrix between the ith token embedding and the hidden state h_i.

This method of encapsulating the representations of all prior hidden states into a single vector has several limitations. For example, lengthy sequences may obscure certain information, not all interdependencies between words may be captured, and the process cannot be parallelized due to the reliance on the computation of all previous hidden states up to the final time step.

Attention mechanisms have been integrated into Seq2Seq models in recent years [9], which has enabled the decoder network to focus on specific parts of the input sequence during decoding and thereby improve the quality of machine translation and other sequence-to-sequence tasks.

2.3 Seq2Seq with Attention

Attention mechanisms have been integrated into Seq2Seq models, enabling them to focus on specific parts of the input sequence when creating outputs, resulting in more precise results. The encoder traditionally produces a single context vector as output, but a new approach involves generating individual output vectors for each hidden state:

$$e^i = [s_i^\intercal h_1, ..., s_i^\intercal h_N]$$

where s_i the decoder hidden state at time step i.

This is achieved by calculating attention scores and converting them into a probability distribution, allowing the decoder to identify the most pertinent hidden states for decoding a particular section of the sentence.

$$\alpha^i = \frac{exp(e^i)}{\sum_j^N exp(e_j^i)}$$

The process is called a softmax transformation, which involves mapping a vector to a probability distribution with values between 0 and 1. This technique is often used in machine learning and deep learning models, especially for natural language processing tasks like language modeling and sentiment analysis.

The attention output is then generated using this probability distribution combined with the decoder's hidden states. This approach has been proven to be effective in Seq2Seq models, particularly in natural language processing tasks such as machine translation, text summarization, and dialogue generation.

$$a_i = \sum_{j=1}^{N} \alpha_j^i h_j$$

This method solved some of the problems with long sequences and dependencies. However, it still had issues with parallelization, which meant that all the time steps had to be computed before the decoder could start. Transformers were introduced in [24] to fix this problem -among others- and resulted in a new era for NLP. They became the foundation for many language models like BERT, ELECTRA, and DeBERTa.

It's worth mentioning that various types of attention mechanisms have been proposed in different studies, including but not limited to Additive Attention (Bahdanau) [5], Multiplicative Attention (Luong Attention) [16], Dot Product Attention, and Multi-Head Attention [24]. In the subsequent section, which delves into the architecture of the Transformer, a comprehensive analysis of the Dot Product Attention and Multi-Head Attention will be presented.

2.4 Transformers

Transformers [24] are advanced models that leverage the attention mechanism to increase training speed and enable parallelization. They have a broad range of applications, such as Auto text memorization, Named Entity Recognition, machine translation and sentiment analysis. The implementation of the transformers is available to the public, allowing individuals to investigate how they optimize parallelization and optimize training time.

2.4.1 Transformer Architecture

The Transformer's encoder is the primary module for all the language representations discussed in this work. Therefore, we will focus solely on the encoder's mechanism of action and disregard the decoder component, as illustrated in Figure 2.

The architecture of the encoder module consists of two primary components: a self-attention layer and a feed-forward neural network. Through experimentation, the authors found that stacking six identical encoders sequentially yielded satisfactory results, similar to the process of hyperparameter tuning. Additionally, the decoder component also incorporates six stacked decoders, though they do not share the same parameters. The input to the encoder first goes through the self-attention layer, allowing the encoder to examine various words in the same input to recognize the relationship between the current word and others (e.g. determining the gender to determine the pronoun). Afterward, the output from the self-attention layer is passed on to the feed-forward network. The bottom encoder contains an embedding layer, and subsequent encoders take the output of the previous encoder as their input.

To calculate the self-attention output, as outlined in [24], the first step is to create three vectors from each of the encoder's input vectors. As such, for each word, a Query vector, a Key vector, and a Value vector will be created by multiplying the embedding by three matrices that were trained during the training process. To do this, the embeddings of each word will be denoted as $x_1, x_2, ..., x_N$. The three trained matrices are the queries

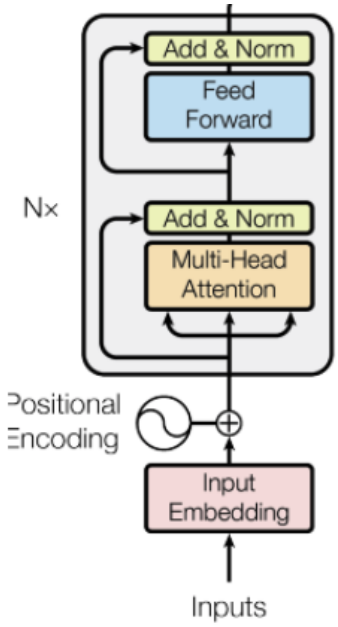


Figure 2: Transformer Encoder from [24]

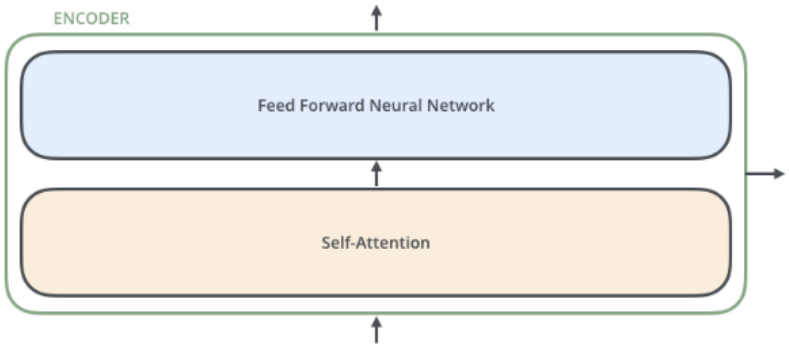


Figure 3: Transformer encoder architecture from [2]

matrix W^Q, the values matrix W^V, and the keys matrix W^K. Then, each of the queries, keys, and values matrices will be multiplied by X to obtain the new matrices Q, K, V, which will be used to calculate the output of the self-attention layer 3.

$$q_i = x_i \times W^Q$$

$$k_i = x_i \times W^K$$

$$v_i = x_i \times W^V$$

Now that we have the q, k, and v matrices prepared, we can begin to score each word against the other words in the same sentence. This will help the model to understand the importance of each word relative to one another when encoding this one. The scoring vector is calculated through taking the dot product of the queries and the keys .

$$e^i = q_i \cdot k_j$$

$$= q_i \times k_j^T$$

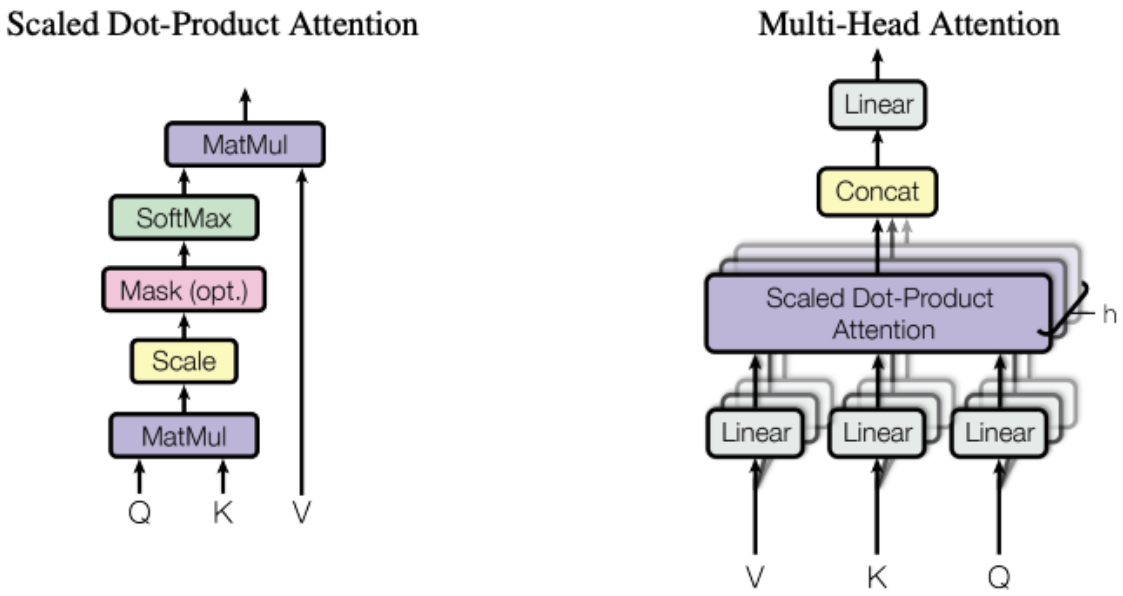


Figure 4: Scaled dot product attention, as well as Multi-Head Attention, explained in Attention is all you need [24]

In their publication [24], the authors presented a novel attention mechanism, known as Scaled dot product attention Figure 4. This approach is similar to Dot-product attention [16], with the exception of a scaling factor of $\frac{1}{\sqrt{d_k}}$, where d_k refers to the dimension of the queries q and keys k. This scaling factor is utilized to obtain more stable gradients, as the magnitude of the dot product might be very large with very large dimensions for the q and k. Now, that we have an overview of all the attention parameters, let's get back to the equations and drive the final attention output.

$$\alpha^i = \textbf{Softmax}\left(\frac{q_i \times k_j^T}{\sqrt{d_k}}\right)$$

Next, The output of the Softmax is multiplied by the values matrix v to give us the final equation for the self-attention layer which will be the input to our feed-forward network. We will denote the output of the attention layer as a.

$$a_i = \textbf{Softmax}\left(\frac{q_i \times k_j^T}{\sqrt{d_k}}\right) v_i$$

The authors of [24] proposed the Multi-Head Attention Mechanism which projects queries, keys, and values multiple times with different, learned linear projections to d_k, d_q and d_v dimensions respectively. This helped the model to focus on different positions, giving the attention layer multiple representation subspaces. To handle the multiple outputs from the attention layers, the authors proposed to concatenate them all into one big matrix Z, and multiply it with a matrix W^O before passing it to the feed-forward network.

Before diving into BERT, I would like to explain two more concepts in the transformer architecture:

- Without time steps like RNNs or LSTMs, how does the transformer encode the position of the sentences?
- What are the Add and Norm layers added after each self-attention layer?

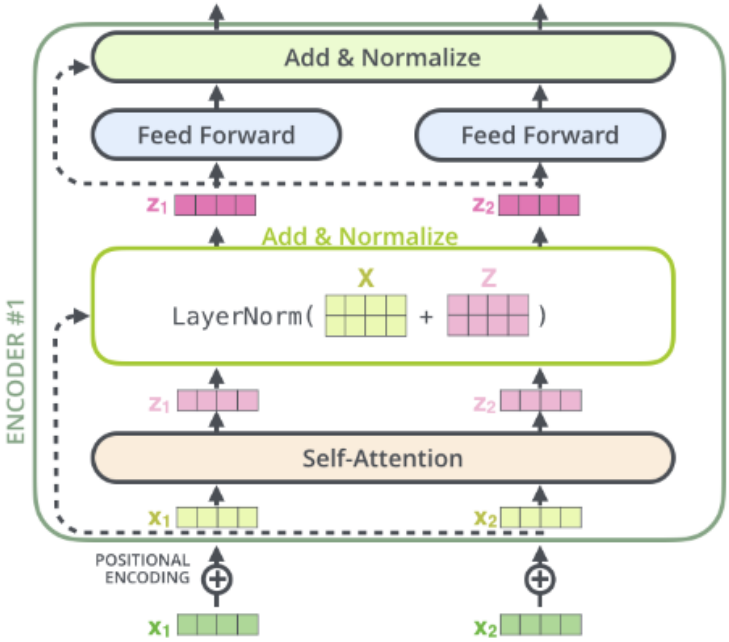


Figure 5: In-depth look for the Encoder from [2]

Previously, when we explained Recurrent Neural Networks (RNNs), we found that we needed to compute all the time steps before the decoder could start, which resulted in a lack of parallelization. Transformers, on the other hand, use positional encoding to add the position of each token to its embedding vector, so the Transformer can understand the order of the words. In the paper, they introduced an approach using sinusoidal methods [24].

The Add and Norm steps are different processes. The Add step is a residual connection, which is a combination of a layer's output with its input and is a solution for vanishing gradients [12].

$$F(X) + X$$

The Norm step is Layer Normalization, which is another normalization technique [4]. This is a computational tool to simplify the training process, improve performance, and reduce training time.

3 Language Representations

In this section, we will cover multiple language representations beginning with BERT, transitioning to DeBERTa which includes a new attention mechanism called Disentangled Attention, followed by ELECTRA which introduces a new way to pre-train a language representation model and finally DeBERTaV3 which builds on the concepts introudced in

ELECTRA. Before we get started, let us review the definitions of some key terms such as pre-training and fine-tuning.

3.1 Pre-training and Fine-tuning

Pre-training involves initially training a model for one task or dataset and then using the parameters or model from the first training to train another model for a different task or dataset. This provides the model with a starting point rather than having to start from the beginning. To illustrate, if we wanted to classify a data set of cats and dogs, we could create a machine learning model, ml, for this classification task. After ml is done training, we would save it along with all its parameters. If we then had another task to accomplish, such as object detection, instead of starting from scratch, we could use ml on the object detection dataset.

Utilizing a fine-tuning approach can be a beneficial way to speed up the development of our model, as it leverages the existing knowledge from a model that is trained on a similar task. However, it is important to recognize that there may be new features that the model needs to learn in order to accurately recognize the new task. For example, a model trained on cars may not have been exposed to truck beds, and so this is something that the model would need to be taught. Nonetheless, there are many features that our model for recognizing trucks can borrow from a model that was initially trained on cars.

3.2 BERT

Google AI introduced BERT as a language representation model in 2018, which is based on the Transformer architecture described in Attention is All You Need. BERT is a bidirectional model, meaning each word can take into account context from both the left and right side of the sentence, whereas the GPT Transformer uses masked self-attention, where tokens can only access the context to their left. BERT has two versions, $BERT_{BASE}$ and $BERT_{LARGE}$. The former has (L=12, H=768, A=12, Total parameters=110M) while the latter has (L=24, H=1024, A=16, Total parameters=340M) as mentioned in [10].

Figure 6: BERT input from [10]

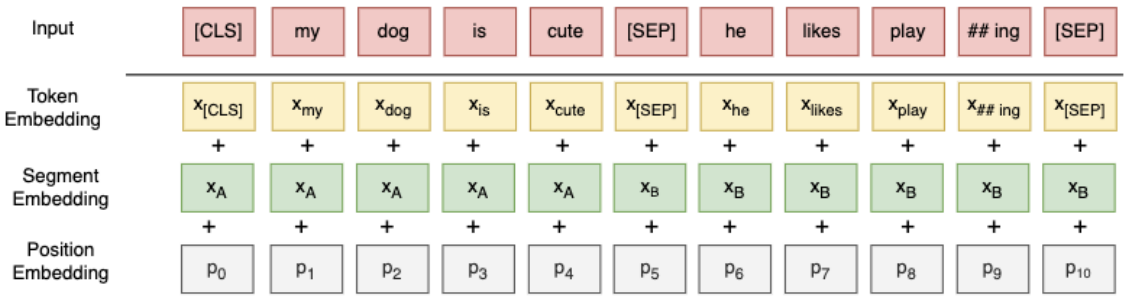


Figure 6 shows BERT's input representation, which consists of the positional embedding, segment embedding and token embedding layer. The positional embedding indicates the position of the word in a sentence, the segment embedding specifies which sentence the word belongs to and the token embedding layer includes special tokens such as [CLS] to mark the start of the sentence and [SEP] to separate two sentences. To obtain the input,

these three embeddings must be summed. The preparation of the data to feed into the embedding layers will be discussed in the training set subsection later.

We highlighted in the transformers how positional encoding is essential to parallelizing the tokens. Although the segment embedding did not show any improvements, we have proposed it as part of our future work.

3.2.1 BERT Pre-training

The pre-training process is split into two different tasks:

- Masked language model (MLM): The task of Masked Language Modeling (MLM) is to mask some tokens and the model must predict the masked token. Training a bidirectional model could be difficult as each word can observe itself and cause the predictions to become too easy. The authors of BERT came up with a way to address this by masking 15% of the words, which is split into three different categories: 80% of the words have the real mask token, 10% are replaced with a random token, and 10% are not changed but flagged for prediction to make sure the model does not know which tokens are replaced and which are unchanged during prediction. The authors also mentioned that leaving some tokens unmasked helped the model performance as none of these tokens will show up during the fine-tuning process.

- Next sentence prediction (NSP). The task was to generate two sentences, sentence A and sentence B, such that 50% of the time when sentence A is presented, sentence B follows and is labeled as isNext = True. The other 50% of the time, a random sentence is presented with the isNext label = false. The length of the combined sentences should sum up to 512, and the sentences do not have to be complete for this task. NSP dropped from all the future works.

3.2.2 Question-Answering

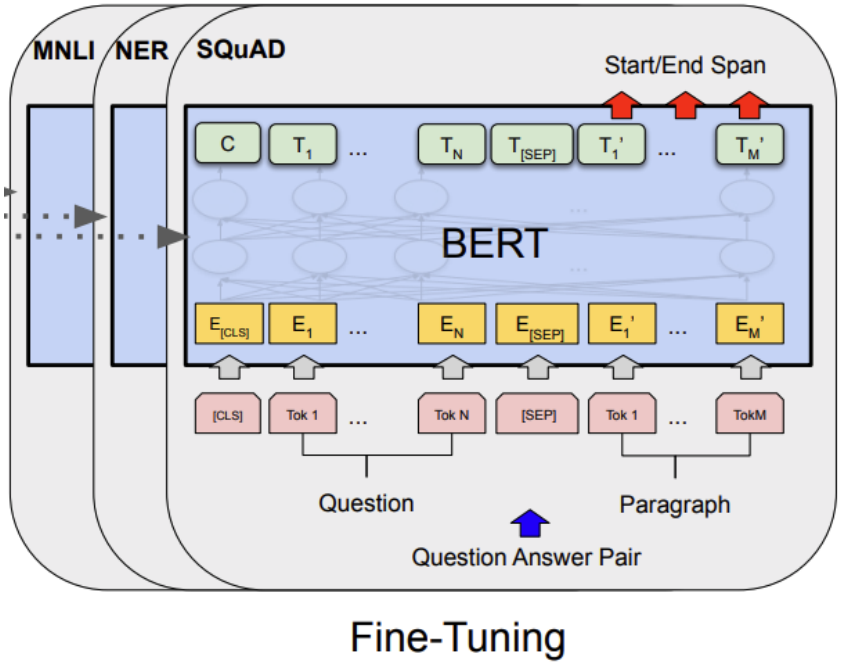
The question-answering process here is distinct from the kind of interactions available with Amazon Alexa and Apple Siri, as it involves providing BERT with a question and a passage from which it must locate the answer. This process is similar to the way Google search highlights the answer when users type in a keyword. We will share results for fine-tuning models over the SQuAD dataset [21]. The SQuAD dataset contains 536 articles with a total of 107,785 question-answer pairs [21]. The model answer is evaluated using two metrics: Exact Match and F1-Score. Exact Match requires the model's answer to match at least one of the ground-truth answers for it to be classified as correct. F1-Score evaluates the accuracy of the predicted answer by calculating the percentage of correctly predicted words, disregarding the order of words (bag of words). In the later experiment section, we will compare the performance of some models over the question-answering task by utilizing these two metrics.

3.2.3 BERT Fine-tuning

The researchers maintained the training settings while experimenting with adjustments to the batch size, learning rate, and number of epochs. Specifically, the batch sizes of 16 and 32, learning rates of 5e-5, 3e-5, and 2e-5, and epoch numbers of 2, 3, and 4 were examined. Additionally, the versatility of the Bidirectional Encoder Representations

from Transformers (BERT) model enables fine-tuning for various tasks, such as sentiment analysis, question-answering, and classification.

Figure 7: Fine-tune BERT for question-answering task [10]



Due to the Transformer's self-attention mechanism, fine-tuning BERT for downstream tasks is relatively straightforward [10]. For Question-Answering specifically, the dataset must be pre-processed into three components. These include the transformed Questions, a passage or context with comprehensive coverage of topics, and an identified answer in the context with a starting token as seen in Figure 7. Additionally, the questions and paragraphs are separated by a [SEP] token. The output of the model will be a Start/End Span showing the beginning of the answer inside the passage.

So far, BERT has depicted each token as a context token, which was previously discussed in the attention section with regard to keys, queries, and values. DeBERTa [14] language representation model has introduced an alternative technique, known as disentangled attention, which employs two vectors to represent each token: one vector conveys the context of the token, while the second vector conveys the direction of the token.

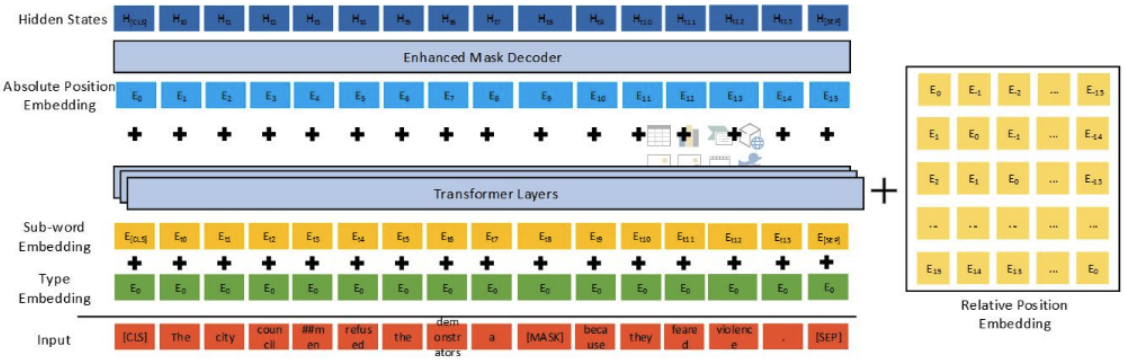
3.3 DeBERTa

The authors of DeBERTa proposed a novel technique, the disentangled self-attention mechanism, that divided the vector into two distinct embeddings: the content and the position embeddings. This approach improved on BERT's representation of the input, as it effectively captures the relative position between words in addition to their content, whereas BERT only considered the positional embedding.

3.3.1 DeBERTa disentangled attention

Previously we used the scoring function e^i to address the q and k vectors and our final scoring equation was:

Figure 8: DeBERTa input from [10]



$$e^i = q_i \cdot k_j$$
$$e^i = q_i \times k_j^T$$

In the present approach, q and v are no longer contained within a single matrix, but instead are distributed into two distinct matrices. Specifically, h represents the content of the current word, while p stands for the relative position of the token. To illustrate the equations, consider the following sequence: "Am I good at playing football?". For any two words selected, the scoring function is now calculated in the following manner, using "Am" at position 0 and "playing" at position 4 as an example:

$$e^{0,4} = \{h_0, p_{0|4}\} \times \{h_4, p_{4|0}\}$$

When we combine the vectors above our equation will as follow:

$$e^{0,4} = h_0 h_4^T + h_0 p_{4|0}^T + p_{0|4} h_4^T + p_{0|4} p_{4|0}^T$$

In this section, each of the four components in the scoring function will be examined in detail:

- The first component, $h_0\ h_4^T$, is the **content-to-content** component. This component is the conventional attention mechanism employed in BERT, wherein the token attempts to identify the nouns in the sentence. For instance, in the sequence "Am I good at playing football?", the word "Am" identifies "I" as the noun in the sentence. In other cases, this component assists the token in deciding the appropriate verb tense, such as "go" or "goes," based on the content of the other tokens.

- The second component, $h_0\ p_{4|0}^T$, is the **content-to-position** component. This component enables each token to comprehend its surroundings. For instance, the token "Am" in the sequence recognizes that it is the initial word in the sentence, which boosts the likelihood of it being a question.

- The third component, $p_{0|4}\ h_4^T$, is the **position-to-content** component. This component facilitates each token in transmitting the appropriate information to the target token. It poses the question, "As the word 'Am' at position zero, what information do I need to relay to the word 'playing' at position 4?"

- The last component, $p_{0|4}\ p^T_{4|0}$, is the **position-to-position** component. This component transmits information from a token at position 0 to a token at position 4. Since the discussion here concerns relative positions, it has been omitted from the equation [10].

Now, after removing the last position-to-position component, we can write a general form of our equations as follow:

$$e^{i,j} = h_i h_j^T + h_i p_{j|i}^T + p_{i|j} h_j^T$$

Like BERT, each vector of q, k, and v has its own projection matrices W_q, W_k, and W_v. However, in DeBERTa, there are two separate matrices for each of the vectors, namely h and p. Therefore, the equations for DeBERTa are as follows:

$$q_c = HW_{q,c}, q_r = pW_{q,r}, k_c = hW_{k,c}, k_r = pW_{k,r}, c_c = hW_{v,c}.$$

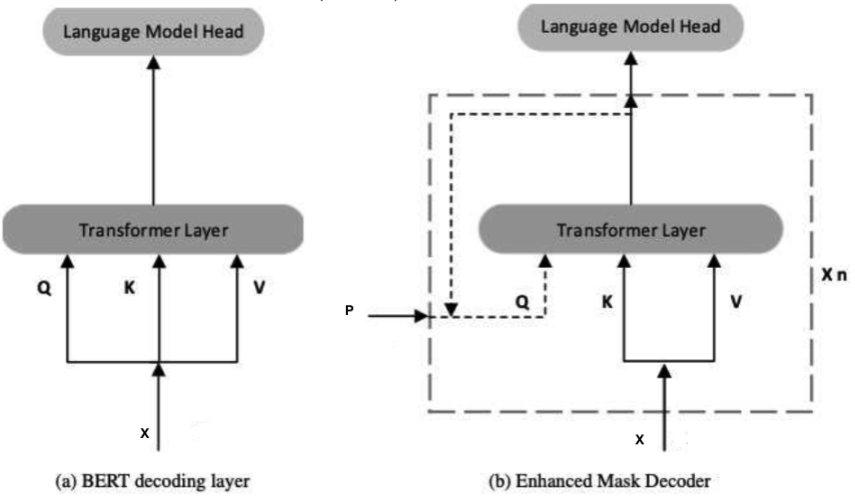
$$e^{i,j} = q^c k^{cT} + q^c k^{rT} + k^c q^{rT}$$

Each of the q^r and k^r is accompanied by a distance measure δ which is defined as follow:

$$\delta(i,j) = \begin{cases} 0 & \textbf{for i - j} \leq \textbf{-N} \\ 2N-1 & \textbf{for i - j} \geq \textbf{N} \\ i-j+N & \textbf{others}. \end{cases}$$

3.3.2 Enhanced Mask Decoder

Figure 9: Enhanced Mask Decoder (EMD) technique compared to BERT MLM from [10]



We have explored the benefits of using relative position encoding in the attention mechanism. Unfortunately, this encoding can also present a challenge; if two words are equidistant from a third word and have the same semantic meaning, the model may struggle to differentiate between them. As an example, consider the sentence "A new [store] opened beside a new [mall]", where both "store" and "mall" are masked for prediction.

Since the words have the same semantic meaning and the same relative distance from the word "new", the model may not be able to accurately distinguish between them.

The authors of DeBERTa introduced the EMD approach to address the issue of words with the same semantic meaning but different relative distances. This approach adds the absolute positional encoding back to the last layer of the encoder, just before the softmax layer. This allows the model to first learn about relative positioning, and then adding the absolute position at the end. Figure **??** shows the difference between BERT and DeBERTa in this regard. If the input sequence P is the same as X and the number of layers X_n is one, then this is similar to BERT.

3.3.3 DeBERTa Pre-training and Fine-tuning

DeBERTa utilizes dynamic masking, a technique that generates a masking pattern for each sequence input to the model, rather than a fixed one. This approach enables training the model for prolonged periods and numerous epochs. The effectiveness of dynamic masking was demonstrated by the authors of DeBERTa, who found that it enhanced the performance of BERT, as illustrated in Table 2.

DeBERTa followed a similar pre-training approach to that of BERT, which involves masking 15% of the words and splitting them into three types: 80% real tokens, 10% replaced with a random token, and 10% left untouched but flagged for prediction purposes. Three different DeBERTa models were developed, namely $DeBERTa_{base}$, $DeBERTa_{Large}$, and $DeBERTa_{1.5B}$. The pre-training time for $DeBERTa_{base}$ was only 10 days, while $DeBERTa_{Large}$ and $DeBERTa_{1.5B}$ took 20 and 30 days, respectively [10].

In line with the methodology employed in BERT, DeBERTa conducted hyper-parameter tuning to determine optimal values for batch size and learning rate. Specifically, they experimented with batch sizes of 16, 32, 48, and 64, and learning rates of 5e-6, 8e-6, 9e-6, and 1e-5. Although the maximum number of epochs used in the fine-tuning stage was 10, the authors did not provide details on the range of epochs explored. For a comprehensive summary of the hyper-parameter tuning performed, please refer to Table 2 (Table 2).

Fine-tuning DeBERTa for Question-Answering (QA) is performed in the same manner as for BERT, as discussed previously in this paper.

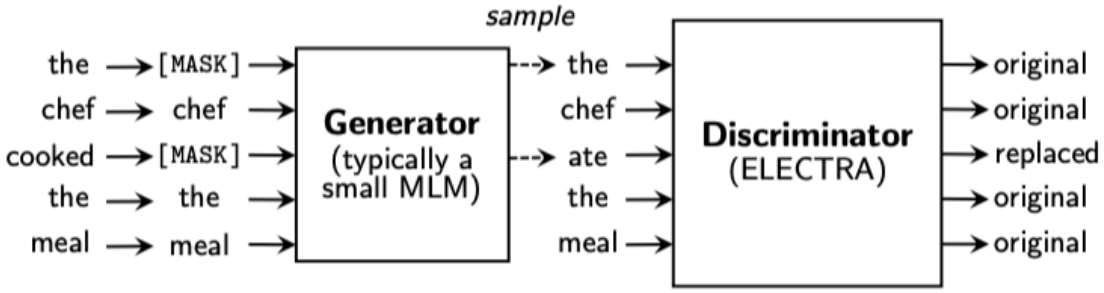
The preceding models, as discussed earlier, involve masking 15% of the tokens, and the model is trained to predict only the masked token. Consequently, these models require significant training time and large corpora to be trained effectively. However, ELECTRA has introduced a novel approach that enables the model to learn from all the tokens, including the unmasked ones, resulting in reduced training time.

3.4 ELECTRA

The issue of insufficient training data, where only 15% of the words are used, has been effectively addressed by ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) [8] through the incorporation of an innovative method known as **Replaced Token Detection (RTD)**. Initially, masking indices are generated in a similar fashion as BERT and DeBERTa, where 15% of the tokens are masked. However, unlike previous methods, the ELECTRA model predicts both the masked and non-masked tokens, with the corresponding indices being saved in a vector designated as m.

$$m = [m_1, m_2, ..., m_k]$$

Figure 10: ELECTRA architecture from [8]



.

where k is the number of tokens to be masked. Subsequently, the tokens that are to be masked will be substituted, resulting in a modified sequence referred to as x^{masked}, which serves as the generator's input Figure 10.

$$x^{masked} = REPLACE(x, m, [MASK])$$

After modifying the sequence of tokens, the generator's role is to predict only the masked tokens. The generator is similar to BERT's model, but it should be smaller in size than the discriminator, which we will discuss later. The loss function for the generator is as follows:

$$P_G(x_i|x^{masked}) = softmax(e(x_i)^\intercal Z_G(x^{masked})_i)$$

where $Z_G(x^{masked})_i$ is the transformer output for a given position i and e is the token embedding. Subsequently, the output of the generator, denoted as $x_{corrupt}$, is obtained and can be interpreted in the following manner:

$$x^{corrupt} = REPLACE(x, m, \hat{x})$$

Subsequent to obtaining $x^{corrupt}$, the discriminator is employed, and here lies the difference between BERT and ELECTRA. The discriminator predicts whether the token is original or previously replaced by the generator (Wrongly predicted by the generator). This way, the discriminator handles all tokens rather than only the masked ones, resulting in a reduced training duration compared to BERT and DeBERTa. The authors mentioned that training ELECTRA small for just 6 hours achieved some good results [8].

The discriminator output can be interpreted as follows:

$$D(x, i) = sigmoid(W^T Z_D(x)_i)$$

where W is the weight matrix, shared between the generator and discriminator. The authors used log-likelihood loss for the discriminator:

$$\mathcal{L}_{Disc} = \left(\sum_{i=1}^{n} -1(x_i^{corrupt} = x_i) \log D(x^{corrupt}, i) \right.$$
$$\left. 1(x_i^{corrupt} \neq x_i) \log(1 - D(x^{corrupt}, i)) \right)$$

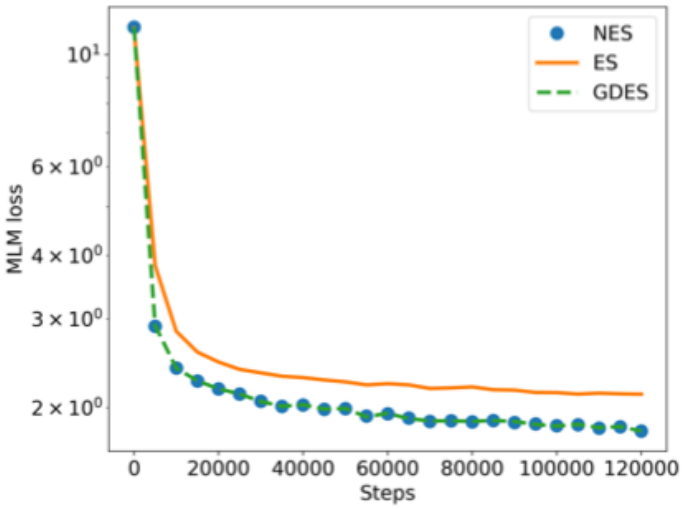
Upon completing the pre-training phase, solely the discriminator will undergo fine-tuning for downstream tasks such as question-answering. The authors emphasized the importance of reducing the generator's size to half or even a quarter of the discriminator's size [8]. This is due to the fact that if a highly efficient generator is employed, which can accurately predict most tokens, the discriminator would encounter fewer instances of replaced tokens.

The training of ELECTRA introduced a novel challenge that only surfaced after the release of DeBERTaV3 [13]. Upon examining the loss functions of both the generator and the discriminator, it became apparent that both aimed to semantically distance tokens from each other, resulting in a tug-of-war dynamics issue that is depicted here [11]. The discriminator tries to differentiate between tokens that are semantically similar, thus pulling their embeddings away from one another. However, the generator and discriminator engage in a tug-of-war game, where each attempts to pull the multi-task learning problem in an opposing direction. This is mainly because the weight matrix W is shared between the generator and the discriminator.

3.5 DeBERTaV3

DeBERTaV3 [13] represents a refinement of the original DeBERTa model, leveraging the advancements introduced in ELECTRA (RTD). Notably, DeBERTaV3 addresses the issue of weight sharing between the generator and the discriminator by introducing a novel technique, gradient-disentangled embedding sharing (GDES) [13]. This is a significant improvement as sharing weights between these components, each of which serves a distinct purpose - the generator for predicting masked tokens and the discriminator for detecting real versus fake tokens - leads to opposing embedding directions and a "tug-of-war" dynamics [13].

Figure 11: MLM training loss of the generator with different word embedding sharing methods from [13]



The authors initially validated their assertions by developing a distinct version of ELECTRA that employed the No Embedding Sharing (NES) method. The NES technique demonstrated improved efficiency of the generator and reduced training time, as presented

in Figure 11. However, when they fine-tuned the discriminator for some downstream tasks, the performance was worse compared to the original ELECTRA that solely utilized embedding sharing (ES). Table 1 shows the performance of the QA task for different techniques.

As depicted in part b of Figure 12, No Embedding Sharing (NES) involves separate updates of the generator and discriminator. Unlike Embedding Sharing (ES), which updates all parameters in a single backward pass, NES updates E_G and θ_G based on $\mathcal{L}_G$, followed by E_D and θ_D through the backward pass involving $\mathcal{L}_{Disc}$ (for the discriminator). As previously noted in ELECTRA, sharing embeddings can lead to improvements in some downstream tasks such as Question-Answering. DeBERTaV3 addresses this by introducing gradient-disentangled embedding sharing (GDES), which allows for weight sharing between the generator and the discriminator, but with certain limitations. Specifically, the generator's gradients are solely calculated based on the generator loss function $\mathcal{L}_G$. One might wonder how this is possible given that GDES involves embedding sharing between the generator and discriminator. The answer is that they share weights only at the end of the training process. Thus, while $\mathcal{L}_G$ updates both E_D and E_G, $\mathcal{L}_{Disc}$ (discriminator loss function) updates only E_D.

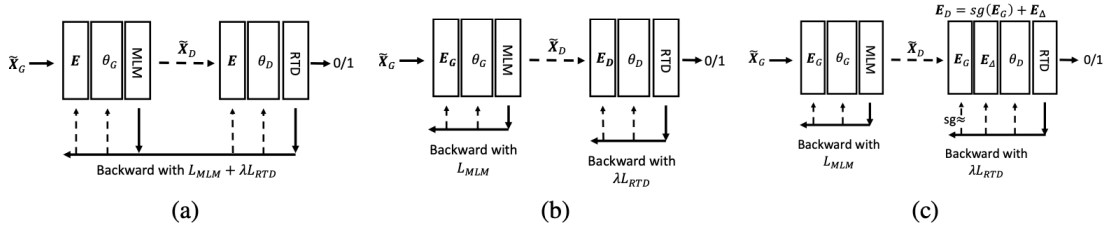
$$E_D = sg(E_G) + E_\Delta.$$

The stopping gradient operator, denoted by sg, is used to restrict the propagation of gradients in DeBERTaV3 [13]. In this approach, E_Δ is initialized as a zero matrix and, for each input sequence, the generator produces a sequence of predicted words by masking some of the original words using a probability function. The discriminator is then used to evaluate the predictions of the generator.

Rather than waiting for the discriminator to complete its forward pass, the gradients for E_G with respect to $\mathcal{L}_G$ are calculated through a backward pass. The discriminator is then run in a forward pass using the output of the generator, and the gradients with respect to the RTD loss are calculated through a backward pass. In this process, the gradients are propagated only through E_Δ [13].

Upon completion of the model training process, E_Δ is added to E_G, and the sum is stored as E_D in the discriminator [13]. This technique helps to ensure that the gradients flow in the right direction and that the generator and discriminator are updated appropriately during training.

Figure 12: differences between different embedding sharing techniques from [13]



.

Table 1: Fine-tuning results on SQuAD v2.0 from [13]

Model (DeBERTa + RTD_{base})	SQuAD v2.0 (F1/EM)
ES	86.3/83.5
NES	85.3/82.7
GDES	87.2/84.5

Table 2: Hyper-parameters for fine-tuning DeBERTa on down-streaming tasks [10].

Hyper-parameter	$DeBERTa_{1.5B}$	$DeBERTa_{Large}$	$DeBERTa_{Base}$
Dropout of task layer	0,0.15,0.3	0,0.1,0.15	0,0.1,0.15
Warmup Steps	50,100,500,1000	50,100,500,1000	50,100,500,1000
Learning Rates	1e-6, 3e-6, 5e-6	5e-6, 8e-6, 9e-6, 1e-5	1.5e-5,2e-5, 3e-5, 4e-5
Batch Size	16,32,64	16,32,48,64	16,32,48,64
Weight Decay	0.01	0.01	-
Maximun Training Epochs	10	10	10
Learning Rate Decay	Linear	Linear	Linear
Adam β_1	0.9	0.9	0.9
Adam β_2	0.999	0.999	0.999
Gradient Clipping	1.0	1.0	1.0

3.6 Experiments

This section will explore the distinctions between DeBERTa, BERT, and ELECTRA models, as well as the advancements made by each model. A comparison of results will be presented to emphasize the individual characteristics of each model, with a specific focus on the question-answering task.

BERT utilizes static masking for token prediction, masking randomly and predicting tokens with a single static mask created during data pre-processing. In order to prevent the repetition of a single mask for every training instance across multiple epochs, a technique was implemented wherein the training data was expanded tenfold, and each sequence was masked in ten distinct ways throughout the 40 training epochs. This means each training

Corpora	$BERT$	$ELECTRA$	$DeBERTa$	$DeBERTaV3$
Wiki+Book (16GB)	✓	✓	✓	✓
OpenWebText (38GB)	-	-	✓	-
Stories (31GB)	-	-	✓	-
Giga5 (16GB)	-	✓	-	-
ClueWeb (19GB)	-	✓	-	-
CommonCrawl(110GB)	-	✓	-	-
CC-NEWS (76GB)	-	-	-	✓

Table 3: DATA - Comparison between pre-training corpora

Table 4: F1 accuracy for static and dynamic masking for BERT on SQuAD 2.0 [15].

Masking	SQuAD 2.0
static	78.3
dynamic	78.7

Table 5: F1 accuracy for static and dynamic masking for BERT on SQuAD 2.0 [15].

Model	SQuAD 2.0 (F1/EM)	Number of params (M)
$BERT_{large}$	81.8/79	345
$ELECTRA_{large}$	-/88.1	335
$DeBERTa_{large}$	91.5/89.0	305
$DeBERTa1.5B$	92.2/89.7	1500

sequence was seen with the same mask four times during training [15]. Conversely, DeBERTa implemented disentangled attention and EMD with dynamic masking (Explained earlier), which improved the model's performance when compared to BERT.

ELECTRA introduced RTD, a novel concept to tackle multi-learning tasks. However, the model faced the challenge of predicting a token and classifying its authenticity simultaneously. To address this, DeBERTaV3 incorporated GDES, an approach that limited weight sharing and excluded the embeddings from both tasks. The efficiency of the algorithm was demonstrated, and further experiments on downstream tasks showed that DeBERTaV3 outperformed all other models, as illustrated in Figure 11. Also, the training data used for each model is illustrated in Table 6.

DeBERTaV3 achieved superior performance in question-answering, as well as in tasks such as Named Entity Recognition, RACE, SWAG, and MNLI, when compared to other models [13]. Furthermore, to fine-tune SQuAD v2.0, sequence tagging tasks were employed, wherein a token classification head was positioned atop the final layer's hidden states of each token.

4 AraBERT (Non-English Language Representation)

The Arabic language is known for its lexical sparsity, arising from its complex concatenative system where many words could give the same meaning in many forms [3]. AraBERT, which is similar to BERT but tailored for the Arabic language, is not as straightforward to fine-tune as BERT is for English, due to the lack of resources in the language [3]. In the following sections, we will discuss the associated challenges and compare AraBERT to mBERT [19], a multilingual version of BERT that supports multiple languages.

AraBERT uses the same configurations used by $BERT_{Base}$ which are, 12 attention heads, 512 maximum sequence length, and a total of 110M params [3]. Also, some extra preprocessing steps had to be taken because of the complexity of the Arabic language, let's start by discussing the pre-training process for AraBERT.

4.1 AraBERT pre-training and Fine-tuning

AraBERT, similar to $BERT_{Base}$, utilizes 12 attention heads, a maximum sequence length of 512, and has a total of 110 million parameters [3]. However, due to the complexity of the Arabic language, additional preprocessing steps were required for AraBERT. In the following sections, we will delve into the pre-training process of AraBERT.

As previously discussed, the complexity of the Arabic language presents a challenge for language modeling due to the presence of words with similar meanings but different forms. for example, a word like "اللغة - Alloga" could be written also as "لغة - loga" because the definite article in Arabic (which is equivalent to the in English) is not an intrinsic part of the word [3]. To address this issue, AraBERT utilizes a segmentation approach by applying Farasa [1] to segment words into stems, followed by training a SentencePiece in uni-gram mode. SentencePiece, developed by Google, is an unsupervised text tokenizer and detokenizer that segments input text into subword units, which are then used as input for NLP models such as AraBERT. Utilizing uni-gram mode, which takes into account only individual characters and not larger units such as bigrams or trigrams, AraBERT is able to effectively capture the morphological complexity of Arabic words. In order to evaluate the effectiveness of this tokenization process, AraBERTv0.1 was introduced, which contains unsegmented words, and compared to the original AraBERTv1 [3].

AraBERT follows a pre-training objective task known as Masked Language Models, which is similar to BERT. During the pre-training process, 15% of the tokens are masked; of those, 80% are replaced with the [MASK] token, 10% are replaced with a random token, and the remaining 10% are kept unchanged.

AraBERT fine-tuned for three different NLP tasks, Sequence Classification, Named Entity recognition, and Question-Answering. I will focus here on Question-Answering in both the fine-tuning and the sections of the experiments.

AraBERT was able to achieve better performance than mBERT in some NLP tasks, including question answering. We can further compare the resources used by each of them; for example, mBERT's usage of tokens was significantly lower than that of AraBERT, which used 24GB of data in comparison to the 4.3GB used by mBERT. Additionally, AraBERT utilized a vocab size of 64k unique Arabic tokens, while mBERT only trained on 2K tokens [6]. AraBERT was only trained on the Arabic language, while mBERT was trained on 104 different languages. Moreover, the preprocessing conducted for AraBERT took into account the complexity of the Arabic language. The pre-training process for AraBERT on QA tasks follows the same process as the one we already discussed for BERT, so we will omit this section here.

4.2 AraBERT and mBERT Experiments

The AraBERT model was fine-tuned for Question-Answering, Named Entity Recognition, and Sequence Classification and resulted in improved performance compared to mBERT. As presented in Table **??** [6], AraBERT was evaluated on the Arabic Reading Comprehension Dataset (ARCD) [17] and outperformed BERT in Sentence-Match and F1 metrics by approximately 2%. The model was trained on the entire Arabic-SQuAD dataset and 50% of the ARCD dataset. However, it did not achieve a higher exact match. The authors of AraBERT discussed that the exact match metric was not effective as the model's answer had to be an exact match of one of the ground-truth sentences. For example, if the question was "Where are the pyramids", AraBERT predicted "Cairo" whereas the

Metric	mBERT	AraBERTv0.1	AraBERTv1
Exact-Match	34.2	30.1	30.6
F1	61.3	61.2	62.7
Sent-Match	90.0	93.0	92.0

Table 6: AraBERT compared to mBERT on Arabic Reading Comprehension Dataset (ARCD)

ground-truth answers had "In Cairo", indicating that the addition of the preposition did not change the meaning but this answer is classified as incorrect.

5 Conclusion

We began by distinguishing between language models and language representations, then discussed some historical language modeling techniques until neural networks and sequence-to-sequence models emerged. We noted that adding attention to seq2seq models improved accuracy and solved issues such as word dependency and long sentences. We then moved on to transformers and self-attention, which were the starting point for many language representation models like BERT and DeBERTa. We can say that transformers marked a new era in NLP and solved the parallelization problems we faced before.

We then discussed BERT, a language representation model, and how bi-directionality helped improve the prediction of masked tokens. We also spoke about pre-training BERT through two different tasks, MLM and NSP. MLM was the main task to pre-train any language representation model we have discussed here, however, NSP did not feature in any of the future works after BERT.

We then moved on to DeBERTa and discussed disentangled attention, and how the relative positional encoding helped achieve better results than BERT. We also noted that having absolute positional encoding was also key, which we demonstrated through the example "A new [store] opened beside a new [mall]". We then moved to ELECTRA and discussed how it takes advantage of the whole dataset and not just the masked token by introducing a new technique called Replaced Token Detection (RTD). We also discussed the "Tug-of-war" problem and how the discriminator and the generator pull the token away from each other.

Then DeBERTaV3 came and solved this problem by introducing a new technique to share the weights between the generator and the discriminator, but with a limit, this technique called Gradient Disentangled Embedding Sharing (GDES). We compared the results for different models and the training data used for each one and how DeBERTaV3 outperformed all the discussed models in the question-answering task. Finally, we moved to a non-English language representation and discussed that the process of training such models is exactly the same as BERT, taking into account the complexity of the selected language. We discussed it for Arabic and compared it to mBERT.

References

[1] Ahmed Abdelali, Kareem Darwish, Nadir Durrani, and Hamdy Mubarak. Farasa: A fast and furious segmenter for Arabic. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 11–16, San Diego, California, June 2016. Association for Computational Linguistics.

[2] Jay Alammar. The illustrated transformer, 2018.

[3] Wissam Antoun, Fady Baly, and Hazem Hajj. Arabert: Transformer-based model for arabic language understanding, 2020.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[6] Ramy Baly, Alaa Khaddaj, Hazem Hajj, Wassim El-Hajj, and Khaled Bashir Shaban. Arsentd-lev: A multi-topic corpus for target-based sentiment analysis in arabic levantine tweets, 2019.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[8] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators, 2020.

[9] Pierre Colombo, Emile Chapuis, Matteo Manica, Emmanuel Vignon, Giovanna Varni, and Chloe Clavel. Guiding attention in sequence-to-sequence models for dialogue act prediction, 2020.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[11] Raia Hadsell, Dushyant Rao, Andrei Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in Cognitive Sciences*, 24:1028–1040, 12 2020.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[13] Pengcheng He, Jianfeng Gao, and Weizhu Chen. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing, 2021.

[14] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention, 2020.

[15] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[16] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015.

[17] Hussein Mozannar, Karl El Hajal, Elie Maamary, and Hazem Hajj. Neural arabic question answering, 2019.

[18] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

[19] Telmo Pires, Eva Schlinger, and Dan Garrette. How multilingual is multilingual bert?, 2019.

[20] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2019.

[21] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[22] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding lstm – a tutorial into long short-term memory recurrent neural networks, 2019.

[23] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.