HAW HAMBURG

MASTER THESIS
Thien Phuc Tran

Global Contextualized Representations: Enhancing Machine Reading Comprehension with Graph Neural Networks

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Thien Phuc Tran

Global Contextualized Representations: Enhancing Machine Reading Comprehension with Graph Neural Networks

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Marina Tropmann-Frick
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 27. Oktober 2023

Thien Phuc Tran

Thema der Arbeit

Global Contextualized Representations:
Enhancing Machine Reading Comprehension with Graph Neural Networks

Stichworte

Natural Language Processing, Transformers, Graph Neural Networks, Knowledge Graphs, Question Answering, Machine Reading Comprehension

Kurzzusammenfassung

In dieser Arbeit werden globale kontextualisierte Repräsentationen (GCoRe) vorgestellt - ein Zusatzmodul zu bestehenden Transformer-basierten Sprachmodellen, das die Leistung bei Question-Answering / Machine-Reading-Comprehension Aufgaben verbessert. Während Transformer-basierte Modelle eine begrenzte Eingabelänge haben und oft nicht in der Lage sind, globalen Kontext und weitreichende Abhängigkeiten zu erfassen, wird GCoRe entwickelt, um diese Probleme zu entschärfen, indem Graph-Inferenz mit Graph Neural Networks auf einem Kontextgraphen durchgeführt wird, der aus dem gesamten Eingabetext konstruiert wurde. Die kontextualisierten Token-Embeddings werden dann mit den aus dem Kontextgraphen abgeleiteten globalen kontextualisierten Merkmalen angereichert. Die Experiment-Ergebnisse zeigen, dass GCoRe das Basismodell im HotpotQA-Datensatz, der aus komplexen Fragen und langen Kontexttexten besteht, um 0,57% übertrifft. Auch im SQuAD 2.0-Datensatz, der kürzere Absätze und Single-Hop-Fragen enthält, übertrifft GCoRe das Basismodell um 0.15%. Besonders bemerkenswert ist, dass GCoRe eine Frage, die Schlussfolgerungen über familiäre Beziehungen erfordert, korrekt beantwortet, während das Basismodell DeBERTa v3 die Frage nicht korrekt beantworten kann.

Thien Phuc Tran

Title of Thesis

Global Contextualized Representations:
Enhancing Machine Reading Comprehension with Graph Neural Networks

Keywords

Natural Language Processing, Transformers, Graph Neural Networks, Knowledge Graphs, Question Answering, Machine Reading Comprehension

Abstract

This thesis introduces Global Contextualized Representations (GCoRe) – an add-on module to existing Transformer-based language models, which helps improve the performance on Question Answering / Machine Reading Comprehension tasks. Whereas Transformer-based models have limited input length and often fail to capture global context and long-range dependencies, GCoRe is designed to mitigate these problems by performing graph inference using Graph Neural Networks on a context graph constructed from the entire input text. The contextualized token embeddings are then augmented with the inferred global contextualized features derived from the context graph. The experimental results demonstrate that GCoRe surpasses the baseline model by 0.57% in the HotpotQA dataset, which comprises complex questions and lengthy context texts. GCoRe also outperforms the baseline model by 0.15% in the SQuAD 2.0 dataset which contains shorter paragraphs and single-hop questions. Notably, GCoRe accurately answers a question that requires reasoning about familial relationships, while the baseline model, DeBERTa v3, fails to answer the question correctly.

Acknowledgements

First and foremost, I would like to extend my deepest appreciation to my professor, Dr. Prof. Marina Tropmann-Frick, for her guidance, encouragement and support throughout my academic journey. I always had the freedom to choose the topics that I am interested in and she was always there to help me when I needed it. She gave me gave me valuable feedback and helped me to improve my work. She is a great and caring person, and I am very lucky to have her as my supervisor.

I am grateful to Silpion IT-Solutions GmbH, especially to my manager, Marc Delling, for his continuous support and understanding over many years. The flexibility and trust he provided during my thesis allowed me to fully concentrate on my research and achieve the best possible results. He is an inspiring person, and I have learned a great deal from him.

I would like to thank my dearest fiance, Chang Truong, for her love and support. She has always been there for me, even when I was not there for her. Her presence in my life is a constant source of inspiration and motivation. I am truly blessed to have her by my side, and I look forward to a future filled with shared dreams and accomplishments.

Lastly, I am deeply thankful to my family and friends. Their encouragement and belief in me have always been the driving force that kept me going.

Contents

List of Figures **viii**

List of Tables **x**

1 Introduction **1**
- 1.1 Question Answering . . . 1
- 1.2 Passage splitting and its problem . . . 2
- 1.3 Objectives . . . 4
- 1.4 Structure . . . 4

2 Transformers **6**
- 2.1 Architecture . . . 6
- 2.2 Self-attention mechanism . . . 8
- 2.3 Multi-head attention . . . 10
- 2.4 Position encoding . . . 11
- 2.5 BERT . . . 12
- 2.6 DeBERTa: Disentangled Attention . . . 14
- 2.7 Limitations . . . 15

3 Graph Neural Networks **17**
- 3.1 Non-Euclidean space and graphs . . . 17
- 3.2 Relational inductive biases . . . 19
- 3.3 Message Passing Neural Networks . . . 20

4 Model architecture **24**
- 4.1 Overview . . . 24
- 4.2 Graph builder . . . 26
  - 4.2.1 Syntactic dependency graph . . . 27
  - 4.2.2 Entity recognition . . . 28
  - 4.2.3 Graph contraction . . . 29

4.2.4 Coreference resolution 29
4.3 Span aggregation block 30
4.3.1 Entity-to-Node Mapping 31
4.3.2 Forward projection and aggregation 31
4.4 Graph attention block 32
4.4.1 Graph inference 32
4.4.2 Backprojection and information enrichment 34
4.5 Training and evaluation 36
4.6 Discussion 38

5 Experiments **40**
5.1 Setup 40
5.2 Results 42
5.3 Interactive experiments 45
5.4 Discussion 49

6 Conclusion **51**
6.1 Summary 51
6.2 Future work 52

Bibliography **53**
Selbstständigkeitserklärung 58

List of Figures

1.1 An example of Question Answering from the SQuADv2 Dataset [21]. . . . 1
1.2 An illustration of passage splitting. 2
1.3 Illustration of an common input sequence layout. 3

2.1 The encoder decoder architecture in seq-to-seq problems. 6
2.2 The transformer architecture [24] 7
2.3 An example of attention mechanism [27] 8
2.4 The scaled dot-product attention mechanism 9
2.5 The multi-head attention mechanism 10
2.6 An illustration of Word2vec and GloVe word embeddings 12
2.7 BERT architecture [7] 13
2.8 An example of an input sequence for BERT [7]. 13
2.9 The difference between BERT and DeBERTa. 15

3.1 An example of a tree where the Euclidean distance between two nodes does not reflect the distance between them. 17
3.2 Euclideanization of data. 18
3.3 An example of a knowledge graph. 19
3.4 Inductive biases in convolutional layers and recurrent layers [3]. 20
3.5 GNN and CNN both exploit the local connectivity of the data [26]. 21
3.6 An illustration of the message passing phase. 22

4.1 The overview of GCoRe's architecture. 25
4.2 The graph builder. 26
4.3 An example of a syntactic dependency graph. 27
4.4 Syntactic dependency graphs of individual sentences are disconnected. . . 28
4.5 An example of entity recognition. 28
4.6 An example of graph contraction. 29
4.7 An example of coreference resolution. 30

4.8 The span aggregation block. 31
4.9 The fordward projection and aggregation. 32
4.10 The graph attention block. 33
4.11 The attention mechanism of GAT [25]. 34
4.12 The computation in a graph layer over multiple time-steps. 34
4.13 Information enrichment. 35
4.14 Forward-projection and Backprojection. 36

5.1 An illustration of the learning rate scheduler. 41
5.2 The trade-off between low and high inject position. 43
5.3 An interactive application for GCoRe. 46
5.4 The context graph for the manually crafted context. 47

List of Tables

5.1 The result (%) for the HotpotQA development set 42
5.2 The result (%) for the HotpotQA development set with different inject positions 43
5.3 The result (%) for the HotpotQA development set with different number of GNN layers 44
5.4 The result (%) for the HotpotQA development set with different dimension sizes for span representations 44
5.5 The result (%) for the SQuADv2 development set 45

1 Introduction

1.1 Question Answering

Question answering (QA) is a crucial aspect of Natural Language Processing (NLP), which aims to develop systems or algorithms capable of responding to questions in natural language. Essentially, QA involves enabling machines to comprehend and extract relevant information from natural language texts like articles or documents.

QA systems are no longer a foreign concept and are widely used in many applications, such as search engines, virtual assistants, information retrieval, or educational systems. In the past decade, big QA systems have emerged, such as Google Duplex [15] and IBM Watson [8]. A state-of-the-art language processing model known as ChatGPT [5, 18] has demonstrated a remarkable ability to generate human-like responses to questions. The appearance of ChatGPT has greatly increased the attention and interest in QA systems.

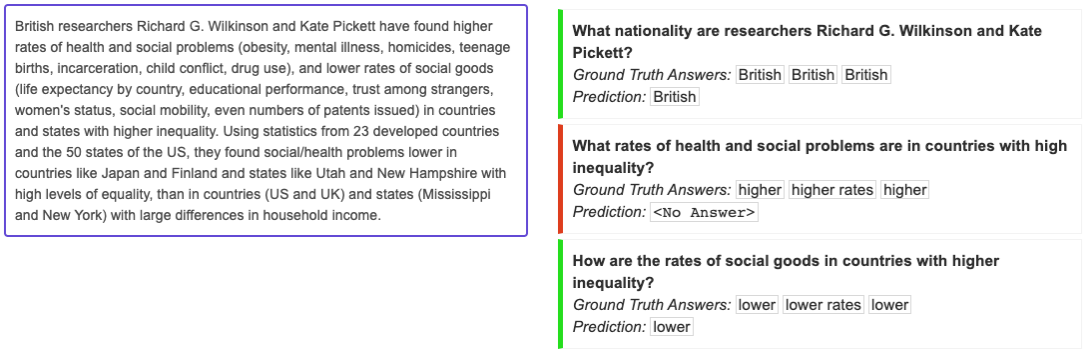

Figure 1.1: An example of Question Answering from the SQuADv2 Dataset [21].

Question Answering requires machines to comprehend text to correctly answer a question. This task is one of the most challenging in Natural Language Processing, as questions

can have answers located anywhere within lengthy contexts. In some cases, reasoning and inference are required to answer the question.

One of the major advancements in NLP is the utilization of transformer-based models. BERT [7] stands out as a prominent transformer-based model, serving as a basis for many state-of-the-art models in the field of NLP. BERT has displayed exceptional performance in various NLP tasks, including Question Answering. Despite its achievements, transformer-based models, in general, still have some limitations, which will be discussed in the next section and Chapter 2.

1.2 Passage splitting and its problem

Analogous to many deep learning models, Transformer-based approaches require a fixed sequence length. However, dealing with lengthy text is often a challenge in question-answering tasks.

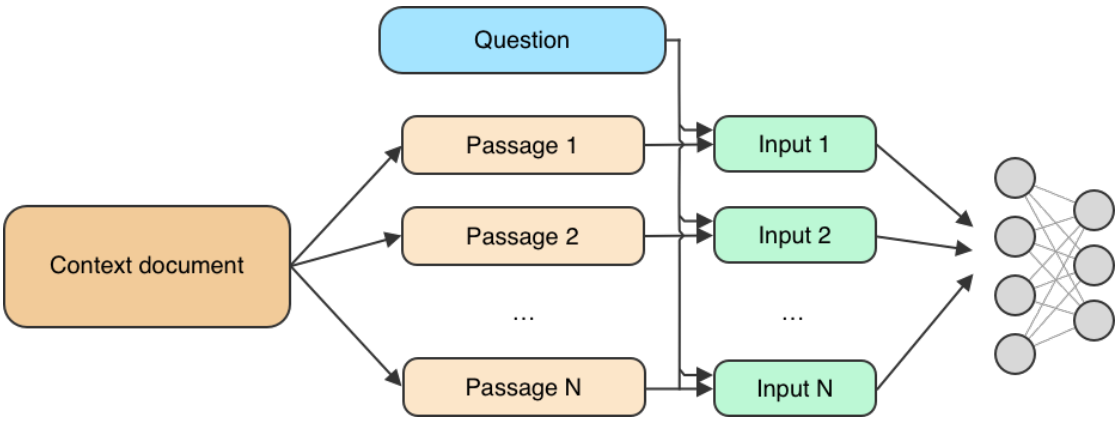

Figure 1.2: An illustration of passage splitting.

For instance, in a QA task, the input text may be a long paragraph or even an entire article. However, BERT-based models usually have a maximum sequence length of 512, which is inadequate to process longer texts. A common workaround is to split a long input text into multiple passages and feed them into the model separately. In cases where the input text is shorter than the maximum sequence length, the text will be padded to the maximum length.

Because there are no dependencies between forward passes, it is necessary to feed the question into the model along with each passage. This allows the model to attend to the relationship between the tokens of the question and the passage. Figure 1.3 provides a visual representation of the typical input sequence layout. Special tokens are added to the input sequence to mark the beginning and end of the question and passage.

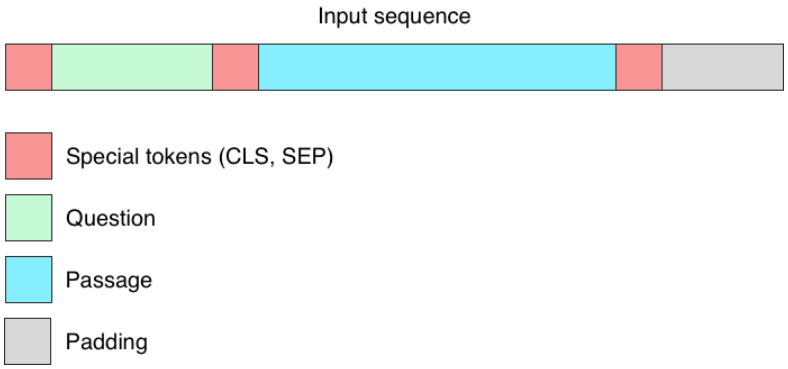

Figure 1.3: Illustration of an common input sequence layout.

Each passage is treated as a stand-alone input sequence; the model is trained to predict the answer for each passage independently. It assigns a score to each token in the passage to indicate the likelihood of the token being the beginning or end of the answer.

As each passage is treated as an individual input sequence and is processed independently, a token can only attend to nearby tokens due to the maximum sequence length limit. Therefore, the model is unable to capture the overall context of the input text and can only comprehend the local context of each passage.

A common workaround is to perform passage splitting with overlapping. In other words, passages are produced by a sliding window of maximum input length with a small stride. By doing so, the model can increase the probability of capturing the necessary information to answer the question in a single passage.

However, if the question is complex and requires information from multiple passages, then the model may struggle to provide a correct answer. Passage splitting not only restricts the model's capability to capture the global context of the input text but also inherently reduces the model's ability to reason. The shortcomings of passage splitting are the motivation for the method proposed in this thesis.

1.3 Objectives

In addition to the limitations of Transformer-based models mentioned in the previous section, the pre-training process requires a large amount of data and computational resources. Such requirements can be a challenge, especially for smaller organizations. With these considerations in mind, the objectives of this thesis are as follows:

- To look for new techniques that can capture global context of input texts without being limited to a fixed input length, so that the model can achieve better performance in Question Answering tasks.
- To develop a method to enhance Question Answering task performance without the need for pre-training. The proposed model should be able to integrate with any pre-trained transformer models.
- To evaluate the effectiveness of the proposed method on different datasets with different characteristics, namely SQuAD 2.0 [21] and HotpotQA [29].

The focus of the thesis is on developing a novel technique that can improve the performance of Transformer-based models in Question Answering tasks. Some of the design choices made in this thesis are influenced by the limitations of time and computational resources. The main objective of this thesis is to provide a proof-of-concept for the proposed method, rather than to achieve state-of-the-art performance on Question Answering tasks.

1.4 Structure

The thesis is structured into six chapters. The first chapter presents a brief introduction to Question Answering. This chapter discusses passage splitting and its limitations, which were the motivation for the thesis. In addition, this chapter presents the objectives and structure of the thesis.

The second chapter presents the background on the Transformer architecture, as well as the building blocks of transformers. The chapter then analyzes the state-of-the-art methods, namely BERT and DeBERTa. The limitations of BERT-based models are also discussed.

The third chapter provides the background for Graph Neural Networks (GNNs). It discusses the nature of non-Euclidean data and the limitations of traditional deep learning methods. The chapter also describes a framework for GNNs, namely Message Passing Neural Networks (MPNNs), which is the fundamental framework for many GNN architectures.

The fourth chapter introduces the Global Contextualized Representations (GCoRe) module, which is the main contribution of the thesis. This chapter first presents an architectural overview of GCoRe and then describes the inner workings of the individual building blocks. The design choices in each building block are also discussed.

The fifth chapter presents the experimental setup and results. The results are presented and analyzed in detail. The effectiveness and limitations of GCoRe are discussed.

The last chapter concludes the thesis and presents future work.

2 Transformers

This chapter provides an objective background on the Transformer architecture along with BERT and its successor, DeBERTa.

2.1 Architecture

Recurrent neural networks (RNNs) have the problem of vanishing gradients, which prevents the model from capturing long-term dependencies. Long short-term memory (LSTM) [13] and gated recurrent unit (GRU) [6] are two widely utilized RNN models that tackle this problem through the adoption of a gating mechanism that allows the model to selectively update the hidden state. Bi-directional LSTM [11] is an extension of LSTM that enables the model to capture the context both before and after a token. Despite their effectiveness, these methods are still challenged by the sequential nature of RNNs, resulting in high computational costs and a limited ability for parallelization.

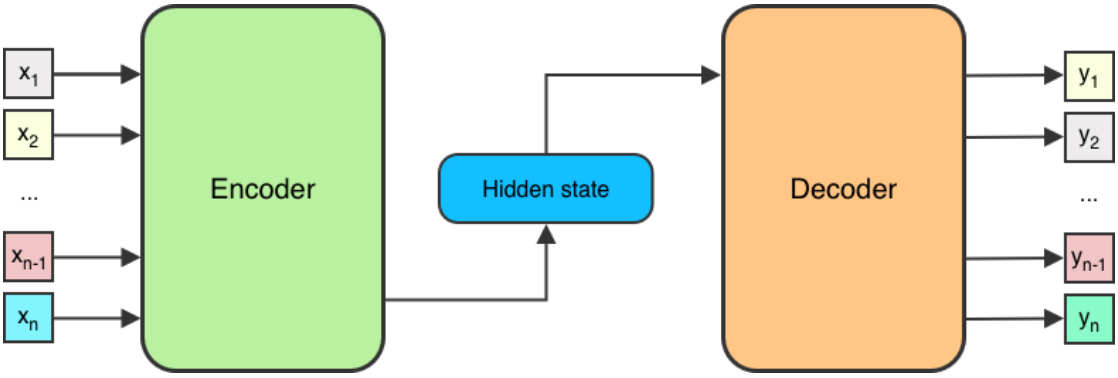

Figure 2.1: The encoder decoder architecture in seq-to-seq problems.

The transformer architecture, presented by Vaswani et al. in the paper "Attention Is All You Need" [24], addressed these limitations by introducing a novel attention mechanism that allows the model to capture long-term dependencies and process the entire sequence in parallel. As demonstrated in various NLP tasks, including machine translation, text summarization, and question answering, the transformer architecture has outperformed prior state-of-the-art models.

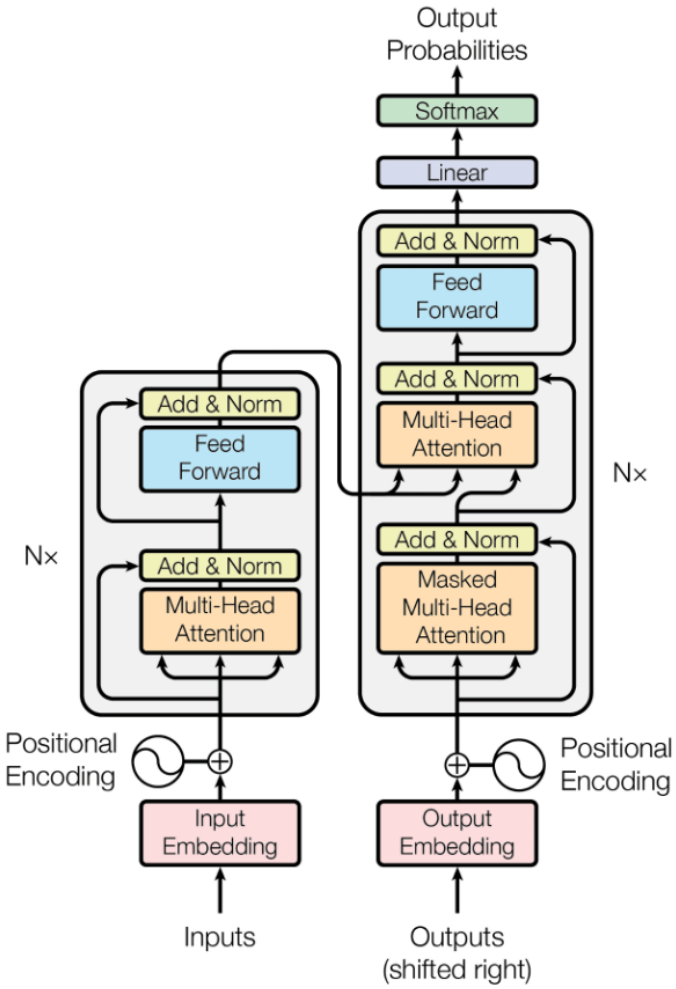

Figure 2.2: The transformer architecture [24]

The transformer architecture features an encoder-decoder structure which was originally proposed by Sutskerver et al. in the paper "Sequence to Sequence Learning with Neural Networks" [23] to solve sequence-to-sequence (seq2seq) problems such as machine translation and text summarization. The encoder accepts the input sequence and encodes it

into a context vector, which serves as a summary of the input sequence. The decoder receives the context vector from the encoder and decodes it into the output sequence. A key difference between the transformer architecture and the original encoder-decoder for seq2seq problems is that the transformer architecture does not use RNNs, but instead uses the attention mechanism which will be discussed in the next section.

A typical transformer model consists of several encoder and decoder layers. The architecture of a transformer is illustrated in Figure 2.2.

2.2 Self-attention mechanism

Self-attention is a mechanism that allows the model to capture the contextual relationships between tokens in the input sequence. The idea of attention was first introduced in the paper "Neural Machine Translation by Jointly Learning to Align and Translate" [1] by Bahdanau et al. for machine translation tasks.

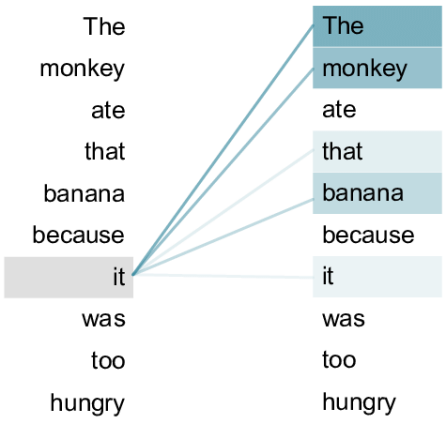

Figure 2.3: An example of attention mechanism [27]

The fundamental concept behind the attention mechanism is to enable the model to selectively focus on important parts of the input sequence. Essentially, implementing an attention mechanism involves computing an attention weight $\alpha_{i,j}$ between token i and j of the input sequence, and subsequently multiplying this attention weight by the corresponding representation vector x_j to obtain the weighted representation vector (see

Equation 2.1).

$$\begin{aligned} \alpha_{i,j} &= attention\_score(x_i, x_j) \\ x'_{i,j} &= x_i + x_j \alpha_{i,j} \end{aligned} \tag{2.1}$$

There are various self-attention mechanisms available. One of the most commonly used self-attention mechanisms is the scaled dot product attention mechanism. Consequently, this thesis will be focusing on the scaled dot-product attention and refer to it as the self-attention mechanism.

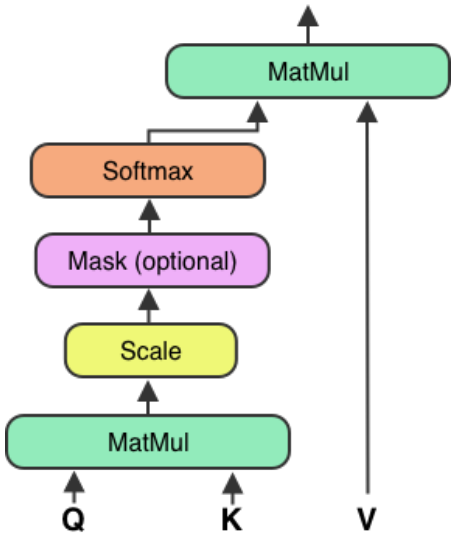

Figure 2.4: The scaled dot-product attention mechanism

Formally, the self-attention for an input sequence X is calculated as follows, where the weight matrices for the query, key, and value are denoted as W_Q, W_K, and W_V respectively, and the dimension of the key is represented by d_k:

$$\begin{aligned} Q &= XW_Q \\ K &= XW_K \\ V &= XW_V \\ Attention(Q, K, V) &= softmax(\frac{QK^T}{\sqrt{d_k}})V \end{aligned} \tag{2.2}$$

The self-attention mechanism computes similarity scores between the query and key by taking their dot product, followed by scaling the result with the square root of the

dimension of the key. The scaled dot-product, represented by $\frac{QK^T}{\sqrt{d_k}}$, is then normalized by a softmax function to obtain the attention weights. The final output of the attention layer is obtained by multiplying the attention weights with the value V. Figure 2.4 provides a technical illustration of the scaled dot-product attention mechanism.

2.3 Multi-head attention

The transformer architecture utilizes the self-attention mechanism multiple times in parallel in one step, each time using different query, key, and value weight matrices. This technique is commonly referred to as multi-head attention. The outputs of the attention heads are concatenated and passed through a linear layer to obtain the final output. The multi-head attention mechanism is visualized in Figure 2.5.

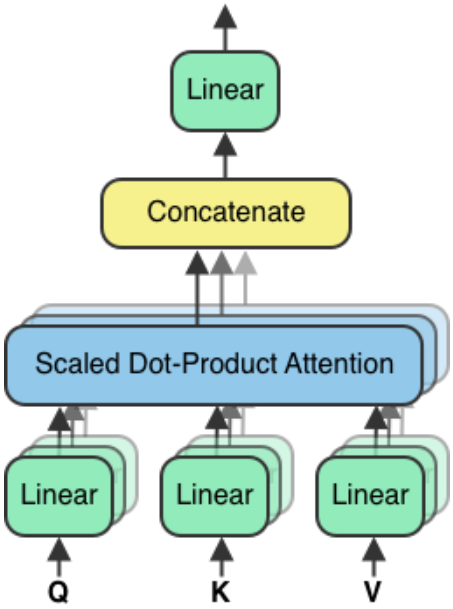

Figure 2.5: The multi-head attention mechanism

Mathematically, the computation for multi-head attention is as follows, where h represents the number of attention heads and W^O represents the weight matrix for the

output:

$$\begin{aligned} MultiHead(Q, K, V) &= Concat(head_1, ..., head_h)W^O \\ head_i &= Attention(Q, K, V) \end{aligned} \tag{2.3}$$

2.4 Position encoding

The Transformer architecture differs from LSTM in that it takes the entire input sequence together and processes it simultaneously, instead of processing input embeddings one at a time sequentially. However, this approach creates a challenge in capturing information related to the position of tokens in the input sequence.

To tackle this issue, the Transformer architecture incorporates position encoding, which introduces positional information into input embeddings. The authors of the paper "Attention Is All You Need" [24] suggest adding position encoding to the input embeddings before model input. The following formula is proposed for computing the position encoding, where *pos* refers to the token position in the input sequence, i represents the embedding dimension, d_E is the embedding dimension size and n is a user-defined constant:

$$\begin{aligned} PE_{(pos,2i)} &= sin(\frac{pos}{n^{2i/d_E}}) \\ PE_{(pos,2i+1)} &= cos(\frac{pos}{n^{2i/d_E}}) \end{aligned} \tag{2.4}$$

The main purpose of the equation 2.4 is to provide a unique encoding for each position. While it is true that both the sine and cosine functions are periodic and collisions may occur, the parameter i varies the frequency for each dimension. This significantly reduces the chance of encoding collisions between positions. Finally, the position encoding is added to the input embeddings to obtain the final input embeddings.

The calculation of position encoding is not limited to using only the sine and cosine functions as illustrated in Equation 2.4. Any function that creates a unique encoding for each position can be used. Numerous techniques have been suggested for computing position encoding, including learned position encoding and relative position encoding.

2.5 BERT

Bi-directional Encoder Representations from Transformers, also known as BERT, is a transformer-based model introduced by Devlin et al. in 2018. Before BERT, the most commonly used models for obtaining word embeddings were Word2vec [16] and Glove [19]. Word2vec and GloVe use a lower dimensional space to represent words instead of relying on one-hot encoding, which becomes high dimensional for large vocabularies.

Word2vec utilizes a shallow neural network to generate word embeddings, while GloVe uses a co-occurrence matrix to generate word embeddings. The resulting embeddings from Word2vec and GloVe are able to capture the semantic and syntactic relationships between words, where similar words tend to be close to each other in the embedding space. Arithmetic operations, such as $e_{king} - e_{man} + e_{woman} \approx e_{queen}$, can be performed with the word embeddings.

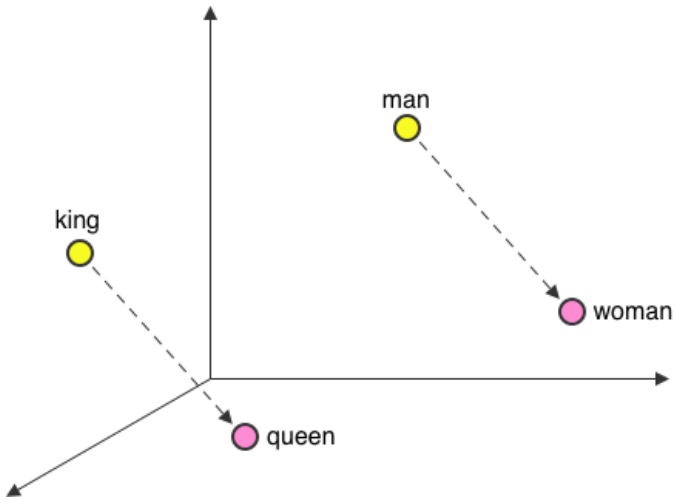

Figure 2.6: An illustration of Word2vec and GloVe word embeddings

However, the word embeddings generated by Word2vec and GloVe are static, meaning they remain the same regardless of the context of the sentence. For example, the word embeddings of the word "cell" are the same in the sentences "The cell is the basic unit of life" and "The prisoner is locked in a cell". This is a key limitation of Word2vec and GloVe, preventing them from capturing the contextual relationships between words.

BERT is a contextualized word embedding model. This implies that the word embeddings vary based on the context. Thanks to the transformer architecture, each token in the input sequence can potentially be influenced by all other tokens in the input sequence;

thus, the word embeddings of the word "cell" in the sentences "The cell is the basic unit of life" and "The prisoner is locked in a cell" are different, because the model can recognize that the word "cell" in the first sentence refers to a biological cell, while the word "cell" in the second sentence refers to a prison cell.

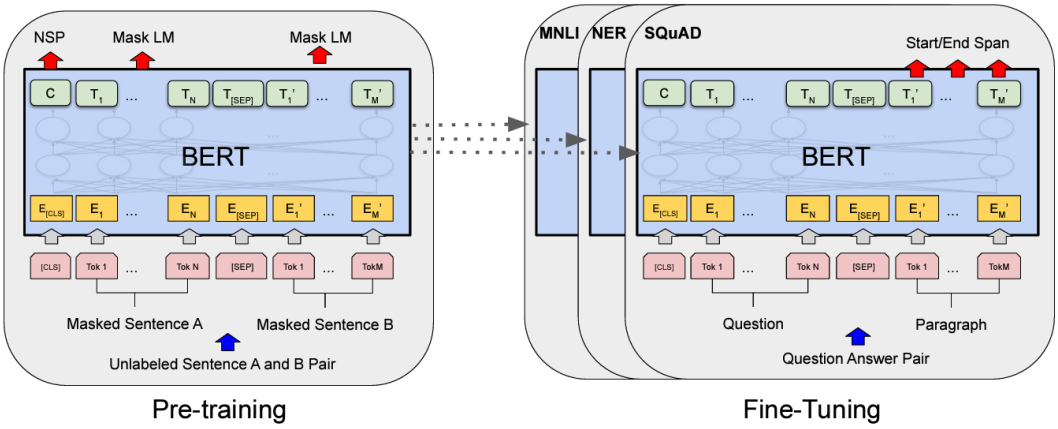

Figure 2.7: BERT architecture [7]

BERT consists of an encoder stack consisting of several transformer layers. Prior to feeding the input sequence into BERT, the input sequence is first tokenized into a sequence of tokens. Each token denotes a word or subword, for instance, "playing" is tokenized into "play" and "##ing" (Figure 2.8). The rationale behind this method is that the model can learn the embeddings of subwords and then combine them to obtain the embeddings of words that are not in the vocabulary.

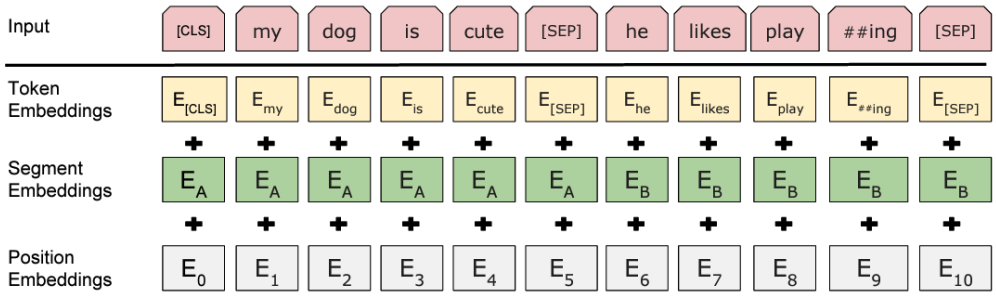

Figure 2.8: An example of an input sequence for BERT [7].

Segment encoding is a binary vector used to distinguish between two segments in the input sequence in Question Answering tasks. The input sequence includes the question and the passage with special tokens indicating the start and end of each segment. The beginning of the input sequence is marked with the [CLS] token to represent the entire input sequence. The [SEP] symbol is appended to the end of each segment to indicate the segment's termination.

The token embeddings are combined with position encoding and segment encoding and then sent through the encoder stack to generate the final output. The result of BERT is a sequence of vectors, each of which represents the embedding of the corresponding token in the input sequence.

The entire output or a part of it can be used as the input for downstream tasks. For instance, in the context of Question Answering tasks, the [CLS] token can serve as classifier input for predicting whether the answer to a question is present in a given passage. Additionally, the output of each token can be fed to a classifier to determine whether the token is the beginning/ending of the answer. Figure 2.8 depicts BERT's input sequence.

2.6 DeBERTa: Disentangled Attention

As a further development of transformer-based architectures, DeBERTa [12] introduces a novel mechanism to improve the performance of BERT-based models – disentangled attention.

The authors of DeBERTa observed that attention to a pair of words depends not only on their content but also on their relative positions. Unlike BERT, which utilizes the sum of content and position embeddings for input, DeBERTa separately utilizes content and position embeddings, meaning that each token is represented by two vectors. The attention score between two tokens is determined by the sum of three components: content-to-content, content-to-position, and position-to-content.

$$\alpha_{i,j} = H_i H_j^T + H_i P_{j|i}^T + P_{i|j} H_j^T \tag{2.5}$$

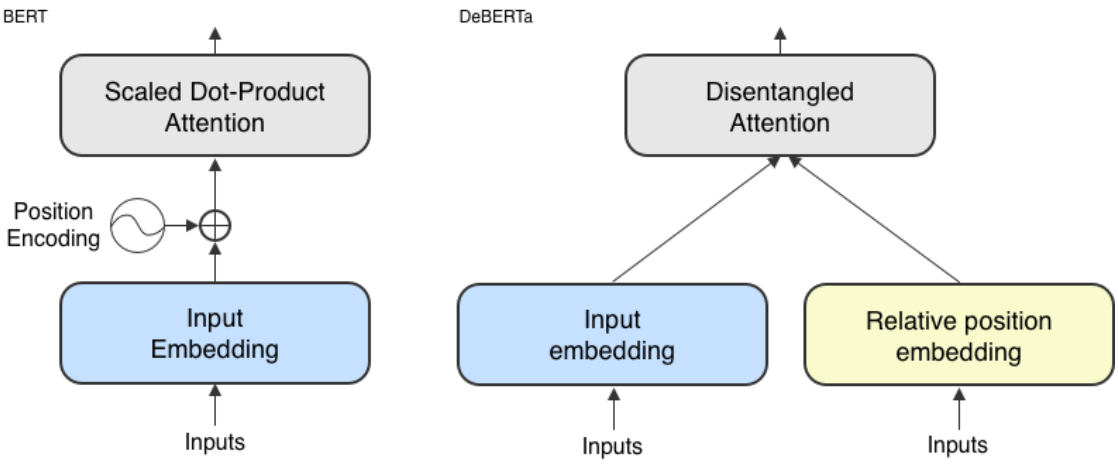

Figure 2.9: The difference between BERT and DeBERTa.

The position-to-position component has been omitted by the authors of DeBERTa because it does not provide any meaningful information. From the perspective of the self-attention mechanism, the disentangled attention can be formulated as follows:

$$
\begin{aligned}
&Q_c = XW_{Q,c} \quad K_c = XW_{K,c} \quad V_c = XW_{V,c} \\
&Q_p = XW_{Q,p} \quad K_p = XW_{K,p} \\
&A_{i,j} = Q_i^c K_j^{c\intercal} + Q_i^c K_{d(i,j)}^{p\intercal} + K_j^c Q_{d(j,i)}^{p\intercal} \\
&Att = softmax(\frac{A}{\sqrt{3d}})V_c
\end{aligned}
\tag{2.6}
$$

Where the function $d(i, j)$ represents the relative distance between tokens i and j in the input sequence. The third component (position-to-content) $K_j^c Q_{d(j,i)}^{p\intercal}$ utilizes the distance $d(j, i)$ instead of $d(i, j)$, as it calculates the attention of the key at position j to the query at position i.

2.7 Limitations

BERT and its variant have achieved remarkable performance in many NLP tasks. Recent advancements, including DeBERTa, have enhanced BERT's performance by introducing

novel attention mechanisms. Referring to Chapter 1, which outlines the challenge of handling lengthy input sequences in BERT-based models, while novel attention mechanisms can help the model to better understand the input sequence, they cannot help the model to capture the global context of long input sequences, as the input sequence is still limited to the maximum sequence length.

Additionally, the author noted the absence of an overview of the input that serves as a big picture, because BERT-based models treat the input sequence only as a sequence of tokens. The model's ability to understand relationships between entities, therefore, is reduced because the attention mechanism operates at the token level, which means that relationships between entities are not explicitly processed.

The author argues that if the model can have a big picture of the input that does not contain tokens of word pieces, but instead contains entities and their relationships, not only can the model capture the global context of the input, but it can also improve the reasoning ability of the model.

Given their capacity to represent entities and their relationships, graphs are a reasonable candidate for capturing the global context of the input sequence. The following chapter will present Graph Neural Networks, a class of neural networks that are capable of processing graph data.

3 Graph Neural Networks

This chapter presents a brief introduction to graphs and gives an overview of Graph Neural Networks (GNNs) – one of the fundamental building blocks in the proposed method.

3.1 Non-Euclidean space and graphs

Traditional deep learning methods are typically designed to process data in Euclidean space, which is defined as a space with a finite number of dimensions in which the Euclidean postulates apply. The distance between two points in Euclidean space can be calculated using the Euclidean distance. For instance, a 2-dimensional Euclidean space is a plane, and a 3-dimensional Euclidean space is the space in which we live.

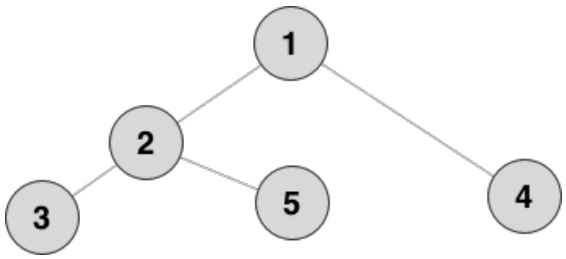

Figure 3.1: An example of a tree where the Euclidean distance between two nodes does not reflect the distance between them.

However, the linear structure of Euclidean space is not always sufficient to represent data. A tree is an example of data that cannot be represented in Euclidean space because the distance between two nodes in the data is not defined by the Euclidean distance, but instead by the number of edges in the shortest path between them. For instance, the distance from node 1 to node 5 in Figure 3.1 is greater than the distance from node 1

to node 4, even though node 5 is closer to node 1 than node 4 in terms of Euclidean distance.

Bronstein et al. [4] point out that many real-world data are not Euclidean; social networks, sensor networks, citation networks, knowledge graphs, and protein-protein interaction networks are all non-Euclidean.

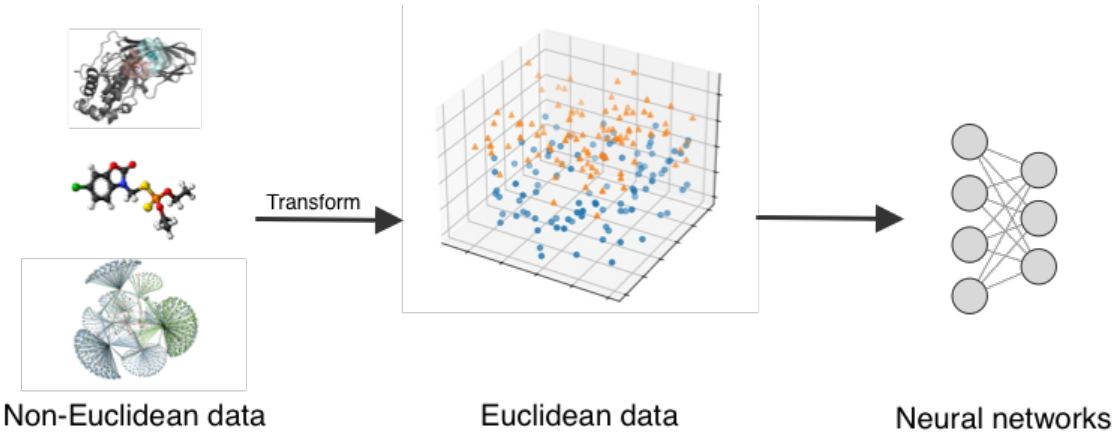

Figure 3.2: Euclideanization of data.

Natural language text is also non-Euclidean. A text document can be represented as a series of words that can be transformed into vectors by using word embeddings. However, this representation loses the relationships between words and the syntactical structure of the text, which are defined by language. The burden of recovering these relationships and structures is shifted to the model.

As a method to make the knowledge embedded in textual data explicit, knowledge graphs (or semantic networks) are introduced. These graphs represent knowledge in a structured way. In Figure 3.3, the knowledge graph depicts the fact that "Paris is the capital of France" and "France is in Europe". The explicit representation of knowledge in knowledge graphs makes it possible to improve or simplify the information retrieval process.

Since conventional deep learning techniques are not capable of processing non-Euclidean data directly, Euclideanization is a widely used workaround in most deep learning tasks. The concept involves transforming non-Euclidean data to Euclidean space before applying conventional methods. Nevertheless, Euclideanization may not always be the optimal

approach, as the data may lose some of its characteristics when converted to Euclidean space. For instance, embeddings can be utilized to transform a knowledge graph into vectors, which can then be processed using conventional deep learning methods. However, this approach results in a loss of the graph's structural information, along with an increased sparsity of the data and thus an increased computational cost.

The need arises, therefore, for deep learning methods capable of handling non-Euclidean data directly without requiring Euclideanization.

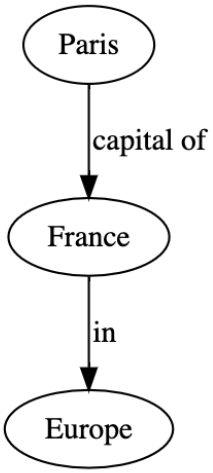

Figure 3.3: An example of a knowledge graph.

3.2 Relational inductive biases

Inductive bias, also known as learning bias, refers to a collection of presumptions or preexisting knowledge about data or the solution space that directs the learning process towards optimal outcomes [3]. In other words, inductive bias decides how a model favors one solution over another.

Geman et al. discussed inductive bias as a tradeoff between bias and variance [9]. A model with a strong inductive bias is more likely to produce the correct result, but it may not generalize well.

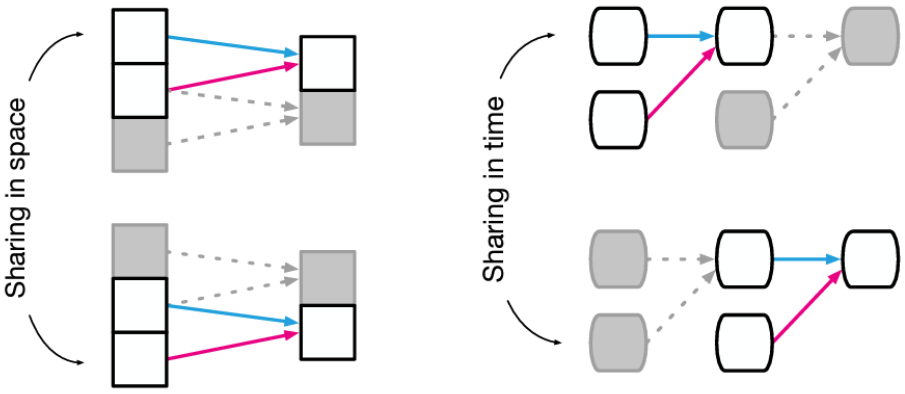

Figure 3.4: Inductive biases in convolutional layers and recurrent layers [3].

Relational inductive biases constrain the relationships between entities and are present in the standard components of deep learning, including convolutional and recurrent layers.

Convolutional layers assume local connectivity in the data and use small kernels (filters) to extract local patterns from the input. Another aspect of relational inductive bias in convolutional layers is translational invariance. Translational invariance means that convolutional layers recognize patterns regardless of their location in the input data by sharing kernel weights. In contrast, recurrent layers take into account temporal dependencies in the data. The output of each time step depends not only on the input of that time step but also on the output generated during the previous time step.

While traditional deep learning methods contain various relational inductive biases, no building block can operate directly on non-Euclidean data, such as graphs, and exploit their structural information. This is the motivation for the development of Graph Neural Networks (GNNs), which will be discussed in the next section.

3.3 Message Passing Neural Networks

Since conventional neural networks struggle to efficiently process graphs due to their non-Euclidean properties, a novel paradigm called Graph Neural Networks (GNNs) has been developed to address this problem. GNNs constitute a category of deep learning models that can process graph data directly without the requirement for Euclideanization.

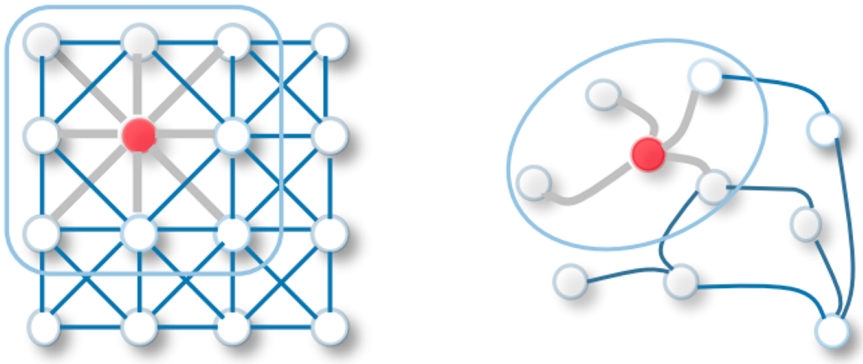

Figure 3.5: GNN and CNN both exploit the local connectivity of the data [26].

Similar to convolutional neural networks, GNNs utilize the local connectivity of data. In the case of images, which are 2-dimensional arrays, local connectivity is defined by the spatial relationship between pixels. For graphs, local connectivity is defined by the relationship between nodes and their neighbors. Figure 3.5 illustrates the similarities between GNNs and CNNs in regard to local connectivity. Essentially, GNNs generalize convolutional neural networks to non-Euclidean data.

Several GNN models have been proposed. They all share the common concept that each node in the graph has a hidden state, whose value is updated based on the node's neighbors and itself. Gilmer et al. [10] introduced the "Message Passing Neural Network" (MPNN), a fundamental framework for GNNs. The MPNN method comprises two stages: message passing and readout.

Message Passing

Let $G = (V, E)$ represent an undirected graph, where V is the set of nodes and E is the set of edges. $v_i \in V$ denotes the i-th node in the graph, while $e_{ij} \in E$ represents the edge linking node v_i and node v_j. The set of neighbors of node v_i is denoted as $N(i)$. Each node v_i has a hidden state h_i^t at timestep t. Moreover, feature vector k_{ij} can be associated with every edge e_{ij}.

Each node v_i computes a message m_i^{t+1} on the basis of its own hidden state h_i^t, the hidden states of its neighbors h_j^t, and the edge features k_{ij}. Equations 3.1 and 3.2 depict the message passing phase of the MPNN framework [10]. A message function M_t and

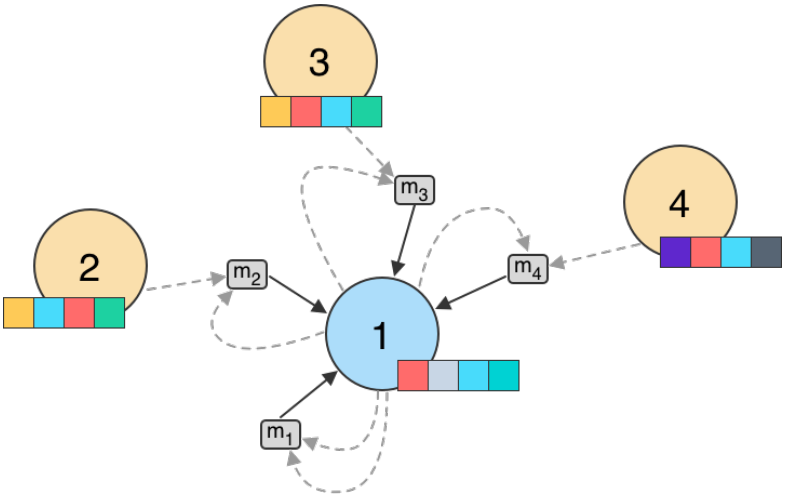

Figure 3.6: An illustration of the message passing phase.

a node update function U_t are used in this process. In a similar way, edge features can also be learned.

$$m_i^{t+1} = \sum_{j \in N(i)} M_t(h_i^t, h_j^t, k_{ij}) \tag{3.1}$$

$$h_i^{t+1} = U_t(h_i^t, m_i^{t+1}) \tag{3.2}$$

Within the MPNN framework, there are various functions available for updating messages and for updating nodes. For instance, Battaglia et al. utilized multi-layer perceptrons (MLPs) for both message and node update functions in Interaction Networks [2]. Meanwhile, Schütt et al. employed a neural network for the message function and a simple sum $U_t(h_i^t, m_i^{t+1}) = h_i^t + m_i^{t+1}$ for the node update function in Deep Tensor Neural Networks [22]. The selection of functions depends on the characteristics of the data and the task at hand.

Readout

After completing the message passing phase, the hidden state of both nodes and edges can be utilized for downstream tasks. In scenarios involving graph-level tasks, a readout

function may be employed to combine the hidden states of the nodes and generate a graph-level representation.

4 Model architecture

This chapter presents an overview of GCoRe, followed by a detailed analysis of the building block mechanisms and a discussion of design decisions.

4.1 Overview

Transformer-based models pose a significant limitation in their fixed input length. A common workaround is to divide the input into smaller chunks (passages) and feed them into the model separately. However, this approach is not ideal as it does not capture the global context of the input. To solve this problem, the author uses Graph Neural Networks to capture the global context of the given text without being limited to the length of the passage. The global context information is then added back to the contextualized features.

Essentially, question answering aims to extract knowledge or information from a given document to find the answer. Such information is often better represented in the form of graphs. Graph data is better suited for logical reasoning than a sequence of tokens because there are explicit relationships between entities on the graph. Furthermore, Graph Neural Networks face no limitations in input length and can execute graph inference on knowledge graphs of any scale. Therefore, the author believes that using graphs and GNNs can help improve the performance on Question Answering tasks.

This thesis introduces Global Contextualized Representations (GCoRe), which can be integrated into any existing transformer-based model. GCoRe is composed of four main components:

1. **Graph Builder** constructs a graph from the text's context. Each node includes metadata for a word or an entity in the text, and each edge represents a relationship between the words.

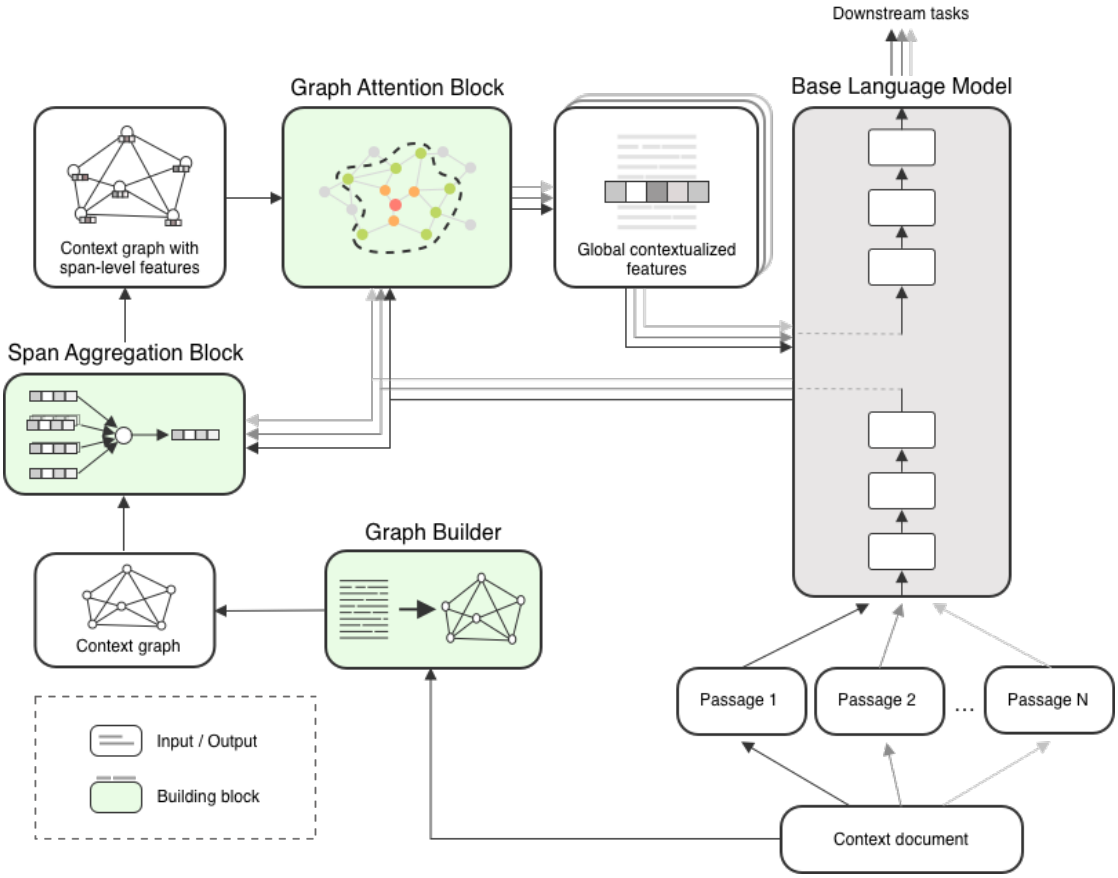

Figure 4.1: The overview of GCoRe's architecture.

2. **Base Language Model** is a Transformer-based model that is used to extract contextualized features from the input text.

3. **Span Aggregation Block** aggregates contextualized features of words in the same span across all passages, creating a single vector representation for each span.

4. **Graph Attention Block**: infers global contextualized features of the nodes in the context graph based on their span representations, and then adds the inferred features to the contextualized features to produce the global contextualized features for individual tokens.

As depicted in Figure 4.1, the input text is divided into several passages, which are fed into the base language model as usual. However, the forward pass of the base language model is interrupted after the first few layers.

A context graph is then constructed by the graph builder utilizing the entire input text. The contextualized features after the first few layers of the base language model are then fed into the Span Aggregation Block along with the context graph, which generates a span representation for each node in the graph.

The Graph Attention Block then infers the global contextualized features from the graph and adds them back to the contextualized features of the individual passages. This process produces the final contextualized features for each passage, which are then consumed by the rest of the base language model before being used in downstream tasks.

The main difference between the proposed approach and traditional methods is that all passages belonging to the same context document must be processed before the outputs for the individual passages can be computed.

4.2 Graph builder

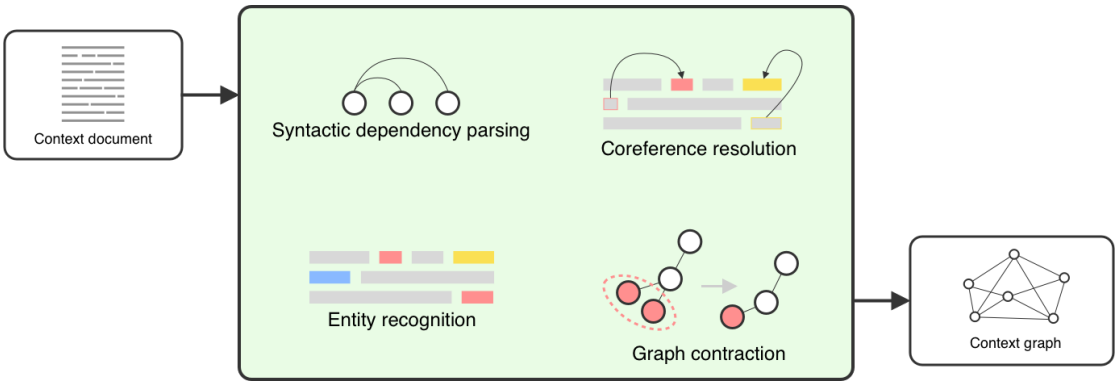

Figure 4.2: The graph builder.

Utilizing context graphs to enhance the ability to capture word dependencies is the central concept of GCoRe. The quality of context graphs, therefore, plays an important role in the performance of GCoRe, both in terms of computational cost and accuracy. The graph builder is responsible for constructing the context graph from the input text.

The graph builder takes text input and outputs a graph with nodes that contain metadata for words or entities, including the word itself, its part-of-speech tag, its named entity tag, and its absolute position in the text. Meanwhile, each edge in the graph indicates a relationship between the words, such as a syntactic dependency or a co-reference.

GCoRe does not limit the construction method of the graph builder. Therefore, any method capable of constructing a graph from the input text may be implemented.

Although knowledge graphs are a suitable choice for the graph builder, their implementation is beyond the scope of this thesis and demands significant resources. To demonstrate the feasibility of the concept and maintain the thesis's scope, the author utilizes a graph builder that combines syntactic dependency parsing, entity recognition, and coreference resolution.

4.2.1 Syntactic dependency graph

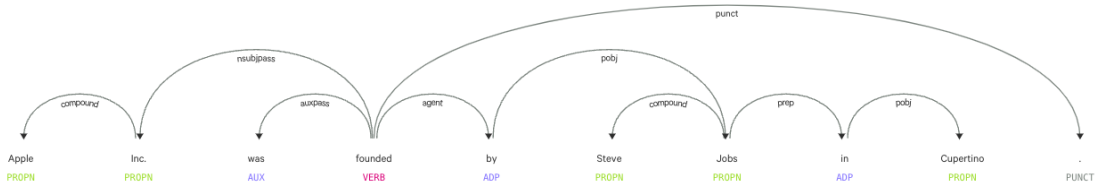

Figure 4.3: An example of a syntactic dependency graph.

Syntactic dependency graphs are used in a variety of NLP tasks, including machine translation, information extraction, and question answering. They serve as fundamental structures that illustrate the grammatical connections between words in a sentence. These graphs depict how words relate to each other, with nodes representing words and edges representing syntactic dependencies, such as subject-verb and noun-modifier relationships. Figure 4.3 shows an example of a syntactic dependency graph.

Although syntactic dependency graphs may not be as expressive as knowledge graphs, they are easier to construct. The author believes that syntactic dependency graphs can be a good starting point for constructing context graphs.

However, one of the problems with syntactic dependency graphs of individual sentences is that they are not connected. This makes the ability of the model to capture the global context of the text more difficult. A solution to this problem will be discussed in later sections.

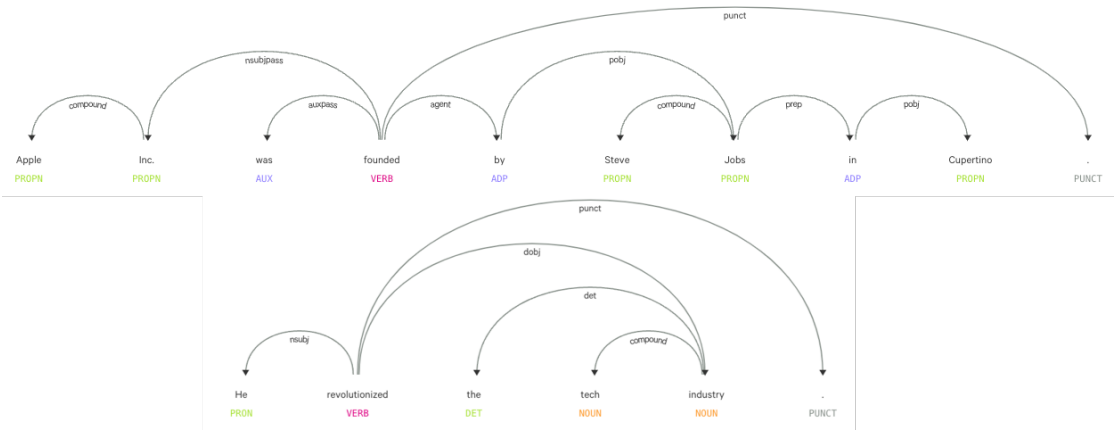

Figure 4.4: Syntactic dependency graphs of individual sentences are disconnected.

4.2.2 Entity recognition

The previous chapter addressed the limitations of BERT-based models and the necessity for a method that can operate on entity level rather than only token level. This thesis used entity recognition to construct word-spans from the input text in preparation for further processing.

Figure 4.5: An example of entity recognition.

Entity recognition tasks require models to identify and categorize specific entities within text, such as names of people, organizations, locations, or dates. This process improves reading comprehension and helps create more structured text. The output of entity recognition is a structured representation of the input, tagged with appropriate labels to identify the entities. For example, in the sentence "Steve Jobs founded Apple Inc. in Cupertino," entity recognition would identify "Apple Inc." as an organization, "Steve Jobs" as a person, and "Cupertino" as a location.

The recognized entities can be used to construct word spans from the input text and lay the foundation for graph contraction and coreference resolution in the later steps.

4.2.3 Graph contraction

Graph contraction is the merging of nodes in a graph. The author suggests a straightforward method of merging nodes that belong to the same word span.

Graph contraction is applied to the syntactic dependency graph to produce a graph that contains word spans instead of individual words. An example of graph contraction applied to the syntactic dependency graph can be seen in Figure 4.6 as depicted in Figure 4.3.

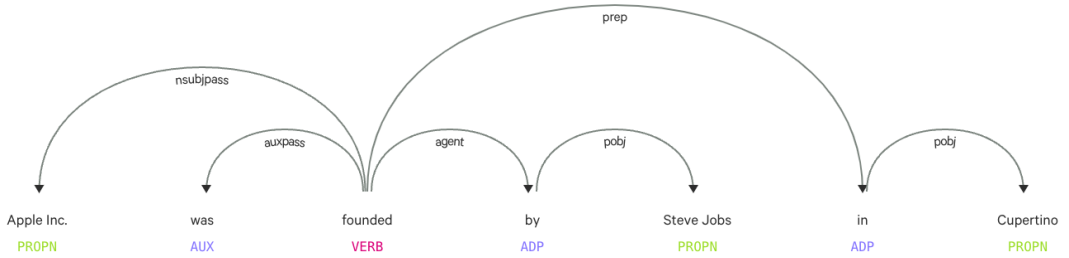

Figure 4.6: An example of graph contraction.

After performing entity recognition and graph contraction, the resulting graph is transformed into an entity-level context graph. Graph contraction reduces the number of nodes in the graph, which lowers the computational cost of the graph inference block. However, the resulting graph is still disconnected, which hinders the model from capturing the global context of the text. To address this problem, the author used coreference resolution.

4.2.4 Coreference resolution

Coreference resolution is the process of identifying all expressions that refer to a specific entity within a text. This step is critical in constructing the context graph because it can connect mentions that relate to the same entity, thereby linking the syntactic dependency graphs of individual sentences.

Consider the previous example followed by a second sentence "Apple Inc. was founded by Steve Jobs in Cupertino. He revolutionized the tech industry." Here, the pronoun "He" lacks explicit identification. Coreference resolution helps to link "He" to its antecedent,

which in this case is "Steve Jobs", and thereby connects the syntactic dependency graphs of the two sentences. The resulting graph is shown in Figure 4.7.

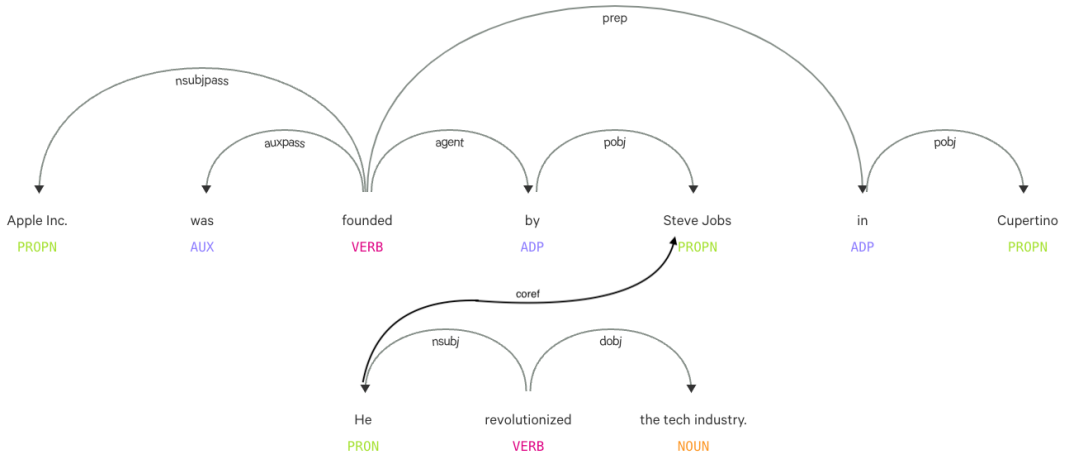

Figure 4.7: An example of coreference resolution.

Combining syntactic dependency parsing, entity recognition, and coreference resolution produces interconnected networks that encapsulate both syntactic and semantic relationships. Entities identified through entity recognition provide informative nodes, while coreference resolution bridges mentions that refer to the same entity.

4.3 Span aggregation block

To perform graph inference on entity level context graphs, it is necessary to modify not only the graph structure but also the representation of the nodes. The Span Aggregation Block (SAB) is responsible for the aggregation of the contextualized features of the words in the same span across all passages, producing a single vector representation for each span.

After the text is split into multiple passages, it undergoes tokenization and is fed into the base language model. The resulting contextualized token embeddings are then passed into the Span Aggregation Block.

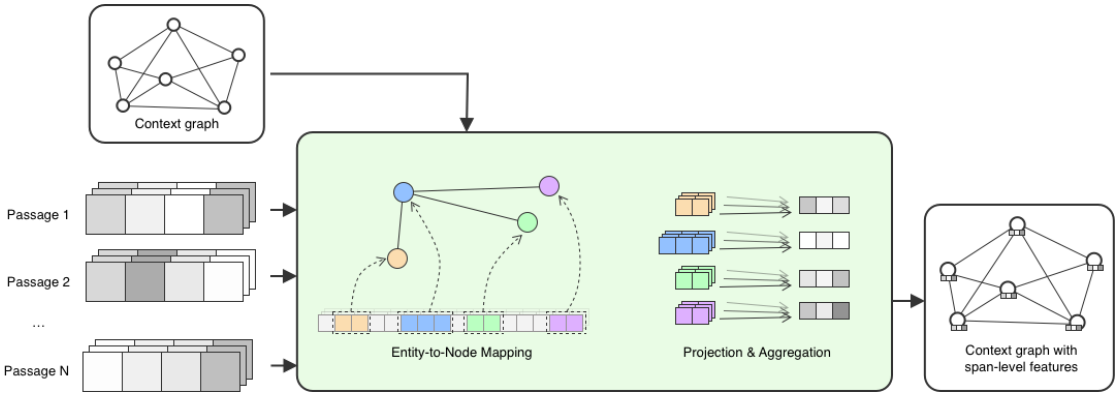

Figure 4.8: The span aggregation block.

4.3.1 Entity-to-Node Mapping

Context graphs in the early stage contain metadata for each node in the graph, which effectively contains information about the start and end positions of each word span.

Using the provided metadata from the context graph, the Span Aggregation Block selects contextualized token embeddings of words within the same span and assigns them to the corresponding node.

Because an entity may appear multiple times within one or more passages, each node within the graph could have multiple sets of contextualized token embeddings linked to it after completing the entity-to-node mapping step.

4.3.2 Forward projection and aggregation

The objective of this process is to merge all contextualized token embeddings associated with a node into a single vector representation for that node.

First, each set of contextualized token embeddings is padded to a maximum length. Next, the padded sets are projected into the same dimensional space as the span representations. This step is referred to as forward projection, to differentiate it from the back projection step in the Graph Attention Block. The forward projection is performed by a linear layer that is shared by all nodes.

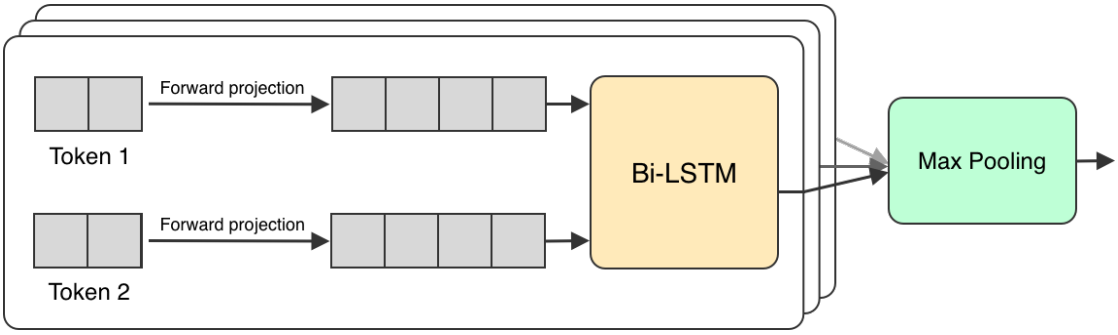

Figure 4.9: The fordward projection and aggregation.

A shared bidirectional LSTM layer aggregates the projected embeddings, resulting in a single vector representation for each set. Each node in the graph may contain multiple span representations after the LSTM layer.

Lastly, a pooling layer is used to aggregate the representations for each node. The author used max pooling in this research since it ensures that the representation of a span is not influenced by the number of times the span appears in the text, but only by its content. The max pooling guides the model to produce positive outputs for important information.

The output of the Span Aggregation Block is a context graph that contains a span representation in each node.

4.4 Graph attention block

The Graph Attention Block (GAB) is the final building block of GCoRe. GAB contains two sub-blocks: graph inference and information enrichment.

4.4.1 Graph inference

The graph inference block takes the span representations from the span aggregation block and infers the contextualized features of the nodes in the graph.

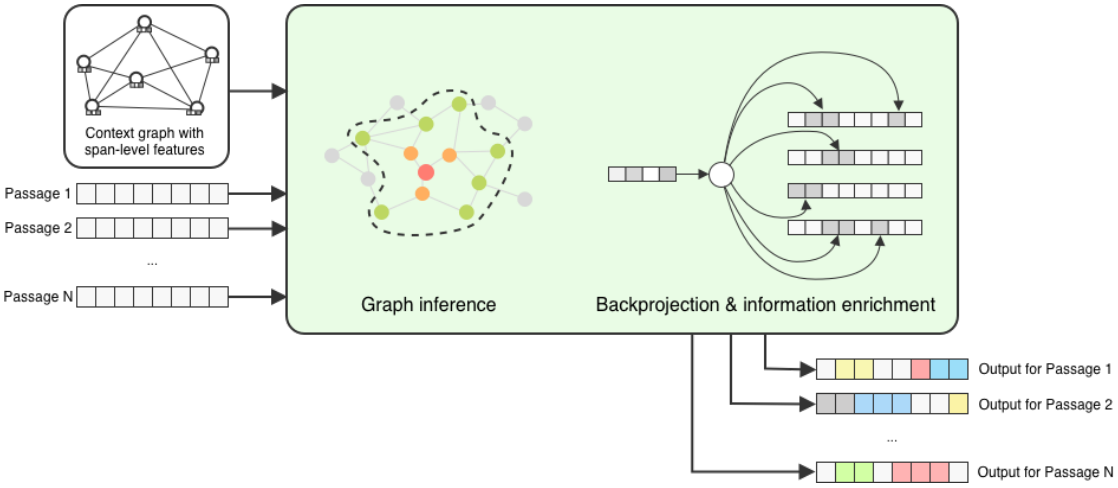

Figure 4.10: The graph attention block.

The graph inference block consists of multiple layers of Graph Attention Networks (GAT) [25] with Jumping Knowledge [28]. The graph inference block works similarly to a transformer, but at entity-level and across all passages of the input text. Each node in the graph inference phase attends to its neighbors and aggregates their features.

The representation h'_i of a node i in a GAT layer is computed as follows:

$$h'_i = \alpha_{i,i} W h_i + \sum_{j \in \mathcal{N}_i} \alpha_{i,j} W h_j \tag{4.1}$$

where $\mathcal{N}_i$ is the set of neighbors of node i, W is a weight matrix, and $\alpha_{i,j}$ is the attention coefficient between node i and j according to paper "Graph Attention Networks" [25].

The author decided to limit the number of layers in the graph inference block to 2-3 layers to avoid the oversmoothing problem in GNNs [17].

Oversmoothing is a problem that arises when a GNN contains too many layers. In each layer, the information of a node is propagated to its neighbors. When the number of layers surpasses a certain threshold, the information is distributed to every node in the graph, resulting in a loss of structural information for the graph.

Edge embeddings are utilized in this work to distinguish the direction of edges, which is important when using syntactic dependency graphs. Although they could differentiate the type of connection between nodes, a more extensive text corpus is necessary to learn

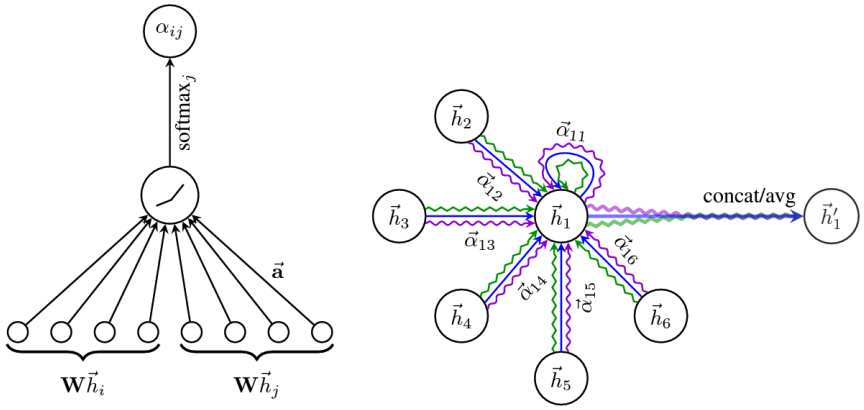

Figure 4.11: The attention mechanism of GAT [25].

the appropriate embedding space. Therefore, this thesis uses a simple edge embedding that distinguishes only the direction of the edges.

The jumping knowledge mechanism aggregates the outputs of all GAT layers in the graph inference block. This approach enables the model to capture information from multiple levels of information propagation in the graph. Some information can be gathered from direct neighbors, while some other information can only be gathered from distant neighbors. This work uses max pooling in the jumping knowledge layer to combine the span representations across all propagation steps.

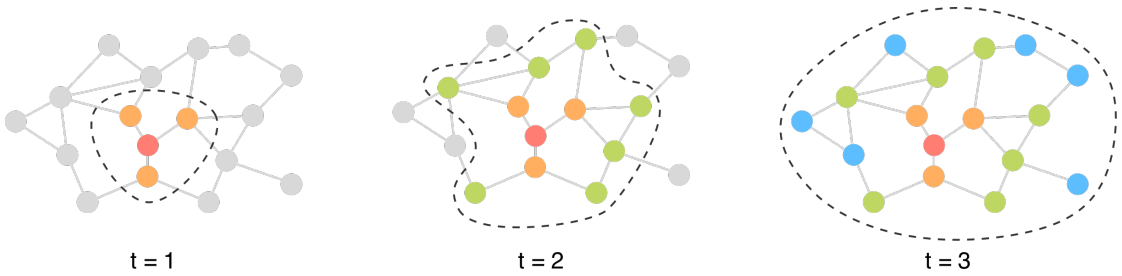

Figure 4.12: The computation in a graph layer over multiple time-steps.

4.4.2 Backprojection and information enrichment

After the graph inference stage, the span representations undergo projection back to the contextualized token embeddings space, getting ready to be added to the contextualized token embeddings originating from the base language model.

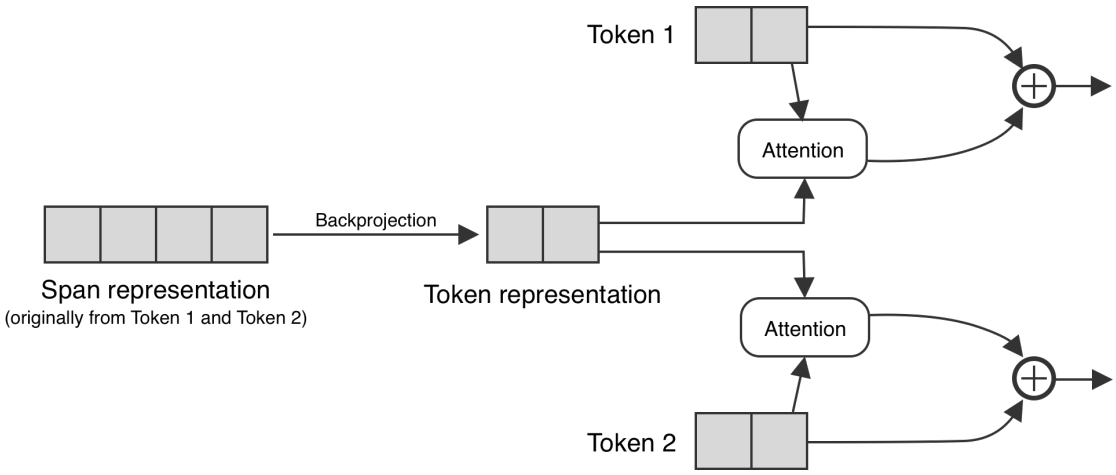

Figure 4.13: Information enrichment.

The information of a span flows back to all tokens belonging to that span. The author implements an attention mechanism to regulate the flow of information from the span representations to their tokens, since some tokens may not be relevant to the actual meaning of the span, such as stop words or punctuation marks.

The span representations only contribute to the tokens that belong to their corresponding spans. This can be achieved by applying a mask to the span representations based on their absolute positions in the text. For every passage, the new token representations are calculated as follows:

$$r_s = W_b h_s \tag{4.2}$$

$$x'_i = x_i + \sum_{s \in S} m_{i,s} Attention(x_i, r_s, r_s) \tag{4.3}$$

where $m_{i,s}$ represents the mask for node i and span s, W_b denotes the weight matrix for the back projection, h_s denotes the representation of span s from the graph inference block, and $Attention(x_i, r_s, r_s)$ refers to the attention between x_i and the back-projected span representation r_s.

The global contextualized representations, then, can be consumed by the rest of the base language model before being consumed by downstream tasks.

Representation alignment

Representation alignment involves ensuring compatibility and alignment of learned embeddings or feature representations. It is intended to allow the different components of the model to co-operate with each other more effectively.

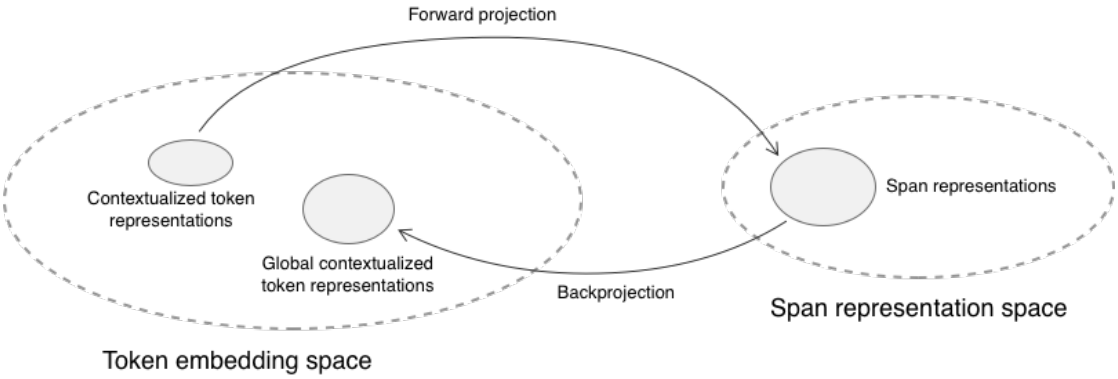

Figure 4.14: Forward-projection and Backprojection.

Figure 4.14 illustrates the process of transforming a representation from one space to another in GCoRe. Initially, the forward projection layer projects the contextualized token embeddings into the space of the span representations. After the graph inference stage, the span representations are projected back to the space of the contextualized token embeddings.

Inspired by KnowBert [20], the weight matrix of the back-projection layer is initialized with the inverse of the weight matrix of the forward projection layer. This initialization encourages the model to learn representations that are compatible with the representations of the base language model.

4.5 Training and evaluation

The typical training approach for transformer-based models processes individual passages separately. The model is trained to predict the answer for each passage, and the final answer is the one with the highest probability. However, this method is not directly applicable to GCoRe, as GCoRe requires all passages to be processed before graph inference can be performed.

To resolve this matter, the author proposes a method that uses memory to store the processed passages of the same sample and performs graph inference after all passages have been processed. This method ensures that the training process for GCoRe is akin to that of standard transformer-based models. As a result, GCoRe can be adapted to existing code bases with minimal modifications.

Algorithm 1 GCoRe's forward procedure

Input: The dataset S which is a non-shuffled list of passage for all samples, the graph builder function *compute_graph* to retrieve the context graph for a sample, the two parts of the base language model M_1 and M_2, the span aggregation block SAB, the graph inference block GAB and the final prediction layer *predict*
Output: The list of predicted answers *answers* for all passages in S

$G \leftarrow null$
$latest\_id \leftarrow null$
$states \leftarrow \{\}$
$answers \leftarrow \{\}$
$S' \leftarrow append(S, null)$
for $s_i \in S'$ **do**
  if $s_i = null$ **or** $latest\_id \neq s_i.sample\_id$ **then**
    if $latest\_id \neq null$ **then**
      $aggregated\_states \leftarrow SAB(G, states)$
      $gcore\_features \leftarrow GAB(G, aggregated\_states, states)$
      for $f_i \in gcore\_features$ **do**
        $a_i \leftarrow predict(M_2(f_i))$ ▷ Predict the answer for each input
        $answers \leftarrow append(answers, a_i)$
      end for
      $states \leftarrow \{\}$ ▷ Clear the memory
    end if
    if $s_i \neq null$ **then**
      $G \leftarrow compute\_graph(s_i.sample)$ ▷ Compute a new graph
      $latest\_id \leftarrow s_i.sample\_id$
    end if
  end if
  if $s_i \neq null$ **then**
    $h_i \leftarrow M_1(s_i)$
    $states \leftarrow append(states, h_i)$ ▷ Store h_i for later processing
  end if
end for
return answers

The process requires that the data remains non-shuffled and be passed to the model in sequential order. As each passage has a sample identifier associated with it, the graph

builder can reuse the computed context graph for all passages that belong to the same sample, helping to reduce the computational cost of constructing context graphs.

The model keeps track of the passages it has processed and the most recent sample identifier in memory. When a new sample is encountered, the model compares its identifier with the stored identifier. If the identifier is different, the model performs graph inference using the data stored in memory and returns the answer. After that, the model clears the memory and proceeds with the current sample. The proposed forward procedure is represented in Algorithm 1.

4.6 Discussion

GCoRe is a versatile add-on module that can be easily integrated into any transformer-based model. One of the significant improvements of GCoRe over the baseline model is its ability to capture the global context of the input text beyond the passage length limitation.

The construction method for the context graph is not limited to any particular method, which allows the model to be flexible and adaptable to different datasets. This work presents a proof of concept by combining syntactic dependency parsing, entity recognition, and coreference resolution to construct the context graph. The author suggests that GCoRe's performance can be further enhanced by employing more sophisticated methods such as a knowledge graph for constructing the context graph. On the other hand, constructing graphs causes more computational overhead. Hence, it is crucial to adopt an appropriate caching strategy to mitigate the computation cost during the training process.

The Span Aggregation Block combines all contextualized token embeddings that belong to a span into a single vector representation. Since entity recognition information is available, performance can be further improved by learning embeddings for entity types. In addition, the Span Aggregation Block can be extended with an external knowledge base to enrich span representations. KnowBert [20] has demonstrated improved model performance through the enrichment of contextualized token embeddings with an external knowledge base.

Edge embeddings can differentiate relationships between entities, potentially enhancing graph inference performance. Creating an effective embedding space requires access to

large text corpora, which is beyond the scope of this work. Nevertheless, in conjunction with the integration of knowledge graphs and external knowledge bases, this is a promising direction for future work.

The training and evaluation procedure of GCoRe requires all passages belonging to the same sample to be processed before graph inference is performed. The author proposed a procedure that uses memory to store the processed passages of the same sample and performs graph inference after all passages of that sample have been processed. The proposed procedure makes the training process of GCoRe similar to that of a traditional transformer-based model, which helps GCoRe adapt to existing code bases with minimal changes.

During training, the procedure requires the training data to be non-shuffled. If distributed training is required, the training data must be split so that all passages belonging to the same sample are in the same batch. The proposed procedure does not differ from the usual procedure during inference, since the model always processes all passages of the sample before returning the answer.

5 Experiments

To evaluate the effectiveness of GCoRe, the author conducts experiments on two datasets: HotpotQA [29] and SQuADv2 [21].

5.1 Setup

The author compares the performance of GCoRe with the baseline model, which is a pre-trained DeBERTa v3 model – a state-of-the-art transformational model for question-answering tasks. In the experiments, the *xsmall* version of DeBERTa v3 is used, with 12 layers and a hidden size of 384. To perform entity recognition and coreference resolution, the spaCy library [14] is utilized along with a pre-trained English language model.

The experiments were conducted using a single NVIDIA RTX 3090 GPU with 24 GB of memory. The models are trained for 8 epochs in all experiments.

Learning rate scheduler

The training procedure is identical for both datasets and is not different from the usual fine-tuning of a transformer-based model, except that the learning rate (LR) is controlled by a modified linear scheduler with warm-up.

The learning rate is increased linearly by the scheduler from zero up to the designated value during the warm-up period. Afterward, the learning rate is then decreased by the scheduler to a pre-defined value instead of zero, which is used until the completion of the training process. The author has found that a capped learning rate is more effective than a learning rate that decreases to zero. The warm-up phase is set to 20% of the total number of training steps. The learning rate scheduler is illustrated in Figure 5.1.

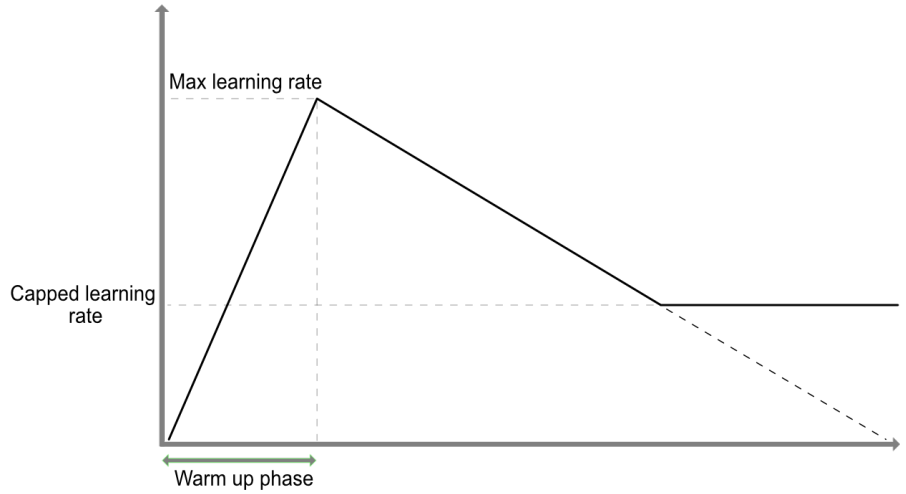

Figure 5.1: An illustration of the learning rate scheduler.

Stride and max entity length

The stride of the sliding window is set to 400. The maximum length of a node in the context graph is 6, meaning that each entity can contain up to 6 tokens, and excessive tokens are truncated.

Dynamic batch size

GCoRe differs from traditional transformer-based models as it requires all passages to be processed prior to graph inference. Hence, the batch size of GCoRe is not static.

During training, the batch size adjusts dynamically based on the number of passages in the training samples. This approach guarantees that all passages belonging to the same sample are in the same batch, and backpropagation occurs after processing all passages and computing the loss. Gradient accumulation aggregates multiple batches' gradients before performing backpropagation. In the experiments, the gradient is accumulated for 8 batches before performing backpropagation.

Classification head

This work adopts a traditional approach to Question Answering by predicting the start and end positions of the answer span. The global contextualized representations are fed

into a classification head, which has a linear layer and a softmax layer, to predict the start and end positions of the answer span.

The linear layer projects the global contextualized representations into a 2-dimensional vector, which is then fed into the softmax layer to compute the probability distribution of the start and end positions of the response span.

5.2 Results

Comparison with baseline model

The first experiment was performed on the HotpotQA dataset, which is a large dataset containing long texts and many multi-hop questions.

The results are presented in Table 5.1. The GCoRe model outperforms the baseline model by 0.57% in Exact Match and 0.25% in F1 score.

Model	Exact Match	F1
Baseline (DeBERTa v3 xsmall)	55.46	69.65
GCoRe + DeBERTa v3 xsmall	**56.03**	**69.90**

Table 5.1: The result (%) for the HotpotQA development set

The result of the first experiment shows that GCoRe is able to improve the performance of the baseline model on the HotpotQA dataset by a relatively large margin, where the text are long and an example usually contains multiple passages.

Inject position

The primary objective of the second experiment is to examine the impact of the inject position on the performance of GCoRe.

The inject position is where to split the base transformer model into two halves and is denoted by the number of the first layers. For instance, the inject position of layer 4 means that the output of the 4th transformer layer will be fed into the GCoRe module.

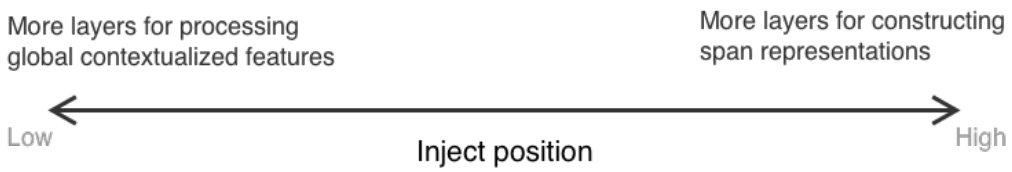

Figure 5.2: The trade-off between low and high inject position.

There is a trade-off between low and high inject position. A high inject position allows the model to capture more context information, as it has access to more layers of the transformer model in the early stage. However, a high inject position also means that the model has fewer layers to process the global contextualized features from the Graph Attention Block.

A low inject position, on the other hand, has more layers to process the global contextualized features from the Graph Attention Block. However, the span representations may contain less information from the context text, since the model has access to fewer layers of the base transformer model in the early stage. This trade-off is illustrated in Figure 5.2.

Position	Exact Match	F1
Layer 4	55.59	69.54
Layer 6	55.63	69.70
Layer 8	**56.03**	**69.90**
Layer 9	55.73	69.65
Layer 10	55.27	69.07

Table 5.2: The result (%) for the HotpotQA development set with different inject positions

The results show that the inject position after the 8th layer yields the best performance.

Number of GNN layers

The author conducted an experiment in which the number of GNN layers was varied. The optimal number of GNN layers depends on the structure of the context graph. For example, hierarchical graphs may require more GNN layers than densely connected graphs.

The table 5.3 shows that the model with 3 GNN layers yields the best performance in this experiment.

Number of layers	Exact Match	F1
2	55.72	69.63
3	**56.03**	**69.90**
4	55.60	69.65

Table 5.3: The result (%) for the HotpotQA development set with different number of GNN layers

Dimension size of span representations

The dimension size for token embeddings utilized was 384, Under the assumption that the dimensions of token embeddings are effectively utilized during the pre-training phase of the base language model, the dimensions of span representations ought to be greater than those of token embeddings.

Dimension size	Exact Match	F1
512	55.53	69.56
768	**56.03**	**69.90**
1024	54.89	68.87

Table 5.4: The result (%) for the HotpotQA development set with different dimension sizes for span representations

The author experimented with span representations using different dimension sizes of 512, 768, and 1024. The results are presented in table 5.4 and show that the model with a dimension size of 768 gives the best performance.

Short context texts in SQuADv2

On the SQuADv2 dataset, the GCoRe model outperforms the baseline model by 0.15% in Exact Match and 0.01% in F1 score, as presented in Table 5.5.

Model	Exact Match	F1
Baseline (DeBERTa v3 xsmall)	80.75	83.88
GCoRe + DeBERTa v3 xsmall	**80.90**	**83.89**

Table 5.5: The result (%) for the SQuADv2 development set

Although the improvement is not significant, the result shows that GCoRe can improve the performance of the baseline model on the SQuADv2 dataset, where input texts are relatively short and an example usually contains only one passage.

This experiment has shown that GCoRe is able to improve the performance of the baseline model even on short context texts. This suggests that the use of graph inference has had a positive impact on token representations. The author hypothesizes that the reason is because GCoRe allows the model to perform entity-level reasoning, which is not possible with the baseline model.

Adding a question node

The author conducted a graph modification experiment involving the addition of a special node to the graph representing the question. The query node is linked to all the nodes in the graph, thereby enabling the model to learn the connection between the query and the entities.

However, the author found that the performance of the model did not benefit from the presence of the question node. The author hypothesized that the entities in the context graph already attend to the question tokens through the self-attention mechanism in the transformer layers. Since the question is represented by a node rather than an entity-level graph, there are no explicit interactions modeled between the entities in the question and the context graph. Consequently, the question node does not provide any additional information to the model.

5.3 Interactive experiments

An interactive application is developed to experiment interactively with the GCoRe model. The application allows the user to enter a question and a context text. The

application, then, provides the answer and a confidence score. A context graph for the input is also displayed.

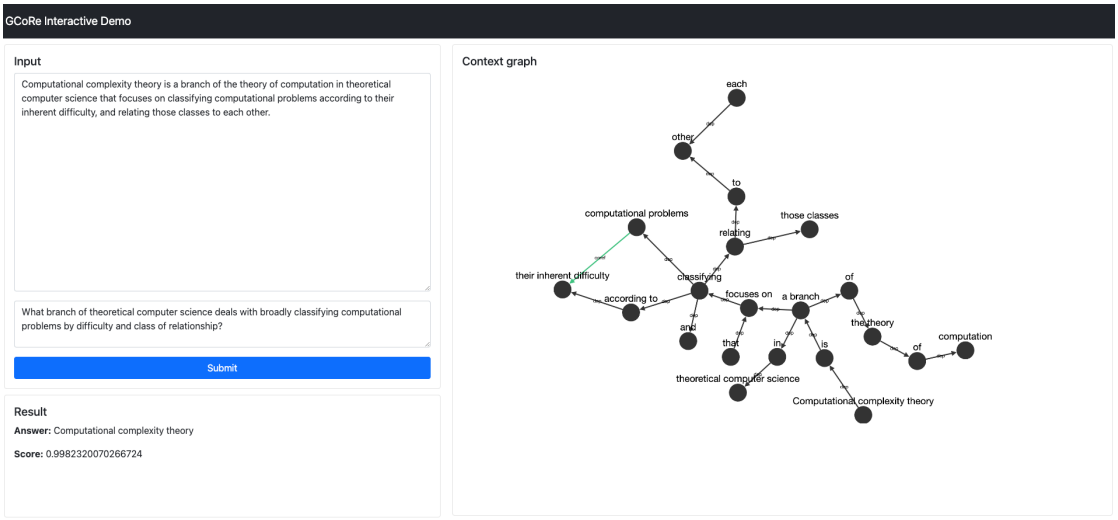

Figure 5.3: An interactive application for GCoRe.

The author employed the GCoRe model, which was trained using the HotpotQA dataset, to conduct the interactive experiments.

Reasoning ability

This experiment utilizes DeBERTa v3's base version as the baseline model, which possesses 86M backbone parameters. The GCoRe model is identical to the model in the previous experiments, utilizing DeBERTa v3's *xsmall* version, which has only 22M backbone parameters.

The question and the context text were crafted to test the reasoning ability of the model. The context describes a familial network where some of the relationships are not explicitly stated. The model, therefore, must perform reasoning on the familial relationships in the context to answer the question accurately.

Context: Alex is a man who is married to Katherine, forming a spousal relationship between them. Katherine, in turn, is connected to Bob as her son-in-law. This intricate familial network continues to extend, as Bob is the father of a daughter. The mother of this daughter is Petty, who shares a sibling bond with a man named Carl.
Question: Who is the grandmother of Bob's daughter?

GCoRe: Katherine	**DeBERTa v3:** Petty

GCoRe was able to answer the question correctly, while the baseline model failed to answer the question. The baseline model predicted the answer to be Petty, who is the mother of Bob's daughter. The baseline model failed to perform reasoning on the familial relationships in the context text.

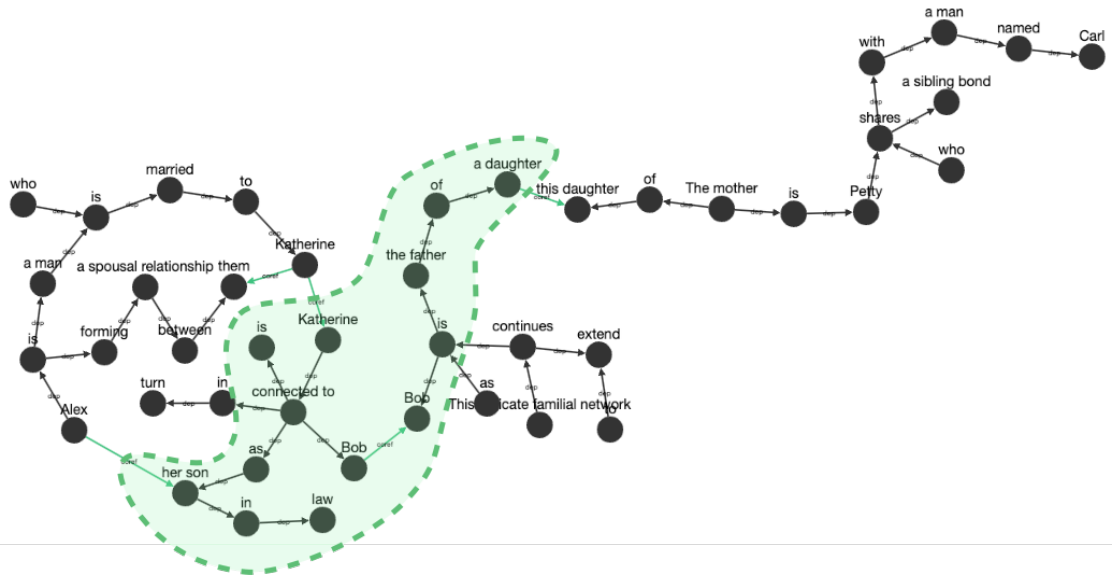

Figure 5.4: The context graph for the manually crafted context.

The context graph for the context text is shown in Figure 5.4. The green area represents the nodes that contribute to the answer. As seen in the green section, coreference resolution linked "Bob" from two subgraphs, connecting the important information from two sentences.

Since the baseline model only applies the attention mechanism at the token level, it is more difficult to capture intricate relationships between entities. On the contrary, GCoRe executes graph inference at the entity level, enabling the model to better comprehend the connections.

Out-of-Domain Question Answering

Context: John Dalton's original atomic hypothesis assumed that all elements were monatomic and that the atoms in compounds would normally have the simplest atomic ratios with respect to one another. For example, Dalton assumed that water's formula was HO, giving the atomic mass of oxygen as 8 times that of hydrogen, instead of the modern value of about 16. In 1805, Joseph Louis Gay-Lussac and Alexander von Humboldt showed that water is formed of two volumes of hydrogen and one volume of oxygen; and by 1811 Amedeo Avogadro had arrived at the correct interpretation of water's composition, based on what is now called Avogadro's law and the assumption of diatomic elemental molecules.

The author utilized data from the SQuADv2 dataset to conduct out-of-domain question answering on the topic Oxygen. This topic is not included in the HotpotQA dataset and comprises numerous domain-specific terms.

Question: What did John Dalton think that all elements were in number present in compounds?
GCoRe: monatomic

Question: What was Dalton's erroneous formula for water?
GCoRe: HO

Question: What element did Gay-Lussac and von Humboldt discover was present in twice the amount of oxygen in water?
GCoRe: diatomic elemental
Ground truth: hydrogen

The model answered some questions correctly. However, it was unable to answer more complex questions that required logical reasoning about the topic.

5.4 Discussion

HotpotQA and SQuADv2 are two different datasets that contain different types of questions. HotpotQA consists of multi-hop questions that require the model to reason over multiple passages. In contrast, SQuADv2 consists of questions that can be answered in a single pass. The performance of GCoRe outperforms the baseline model on both datasets. The improvement is more significant in the HotpotQA dataset, which contains long texts and complex questions with multiple hops. However, the improvement in the SQuADv2 dataset is not significant because the texts are relatively short and the questions are relatively simple. The graph inference mechanism in GCoRe has proven its ability to capture global context more efficiently, which is beneficial for long texts. The baseline model lacks the ability to perform entity-level reasoning, which GCoRe can do.

In this work, a modified linear scheduler with warm up was used to control the learning rate. The original linear scheduler, which linearly decreases the learning rate to zero, did not help improve performance because the learning rate was too low for the model to learn. A small number of training steps during the fine-tuning process results in a steep slope of the learning rate reduction, causing the learning rate to drop to zero too quickly, preventing the model from learning. The capped learning rate, on the other hand, allows the model to learn more effectively while still benefiting from the warm up phase.

Typically, the step size of the sliding window is kept small to ensure that tokens in the passages have a better chance to attend to important information. In the experiments, the stride was set to 400, which is close to the maximum length of context in a passage. The reason is that the context graph already captures the global context, which allows the model to attend to important information more effectively. Therefore, a token doesn't need to attend to too many other tokens in its surroundings.

There is a trade-off in the selection of the inject position. The author conducted experiments with different inject positions and discovered that the inject position after the 8th layer gives the best performance. As the inject position increases, the model has access to more layers of the base transformer model in the early stage, which means that the context graph can receive more information from the context text. The remaining four layers are sufficient to effectively process the global contextualized features from the Graph Attention Block.

Adding a question node to the context graph did not improve the performance of the model. The reason may be that there are no explicit interactions between the entities in the question and the context graph, and the token representations of the context text already attend to the question tokens through the self-attention mechanism in the transformer layers. However, another entity-level graph representing the question, a question graph, might improve the performance of the model. This topic is left for future work.

During the interactive experiments, the author found that GCoRe was able to answer some out-of-domain questions. However, the model was unable to generate accurate answers to more complicated questions.

The model successfully provided the correct answer to a question created by the author that required reasoning about the familial relationships in the context text, while DeBERTa v3 failed to provide the correct answer. GCoRe was able to produce a context graph that linked important information, allowing the model to reason more effectively.

6 Conclusion

6.1 Summary

The pre-training process of a base language model is expensive in terms of computation and time. The major goal of this work is to construct novel methods to improve the performance of existing pre-trained base language models on Question Answering tasks without requiring further pre-training of the base language model.

Transformer-based models struggle to capture long-term dependencies in input data, particularly with lengthy inputs. This problem arises because these models have a fixed input length, forcing inputs to be split into multiple passages, which then need to be processed independently. This can lead to difficulty in capturing relationships between tokens at the beginning and the end of a long text. GCoRe addresses this issue by creating a context graph for the entire input text and using it to perform graph inference. Furthermore, GCoRe can be easily integrated into any pre-trained Transformer-based model with minimal changes, since it only interrupts the forward pass of the base language model and enriches token representations with global contextual information. Span representations in GCoRe are encouraged to adapt to the learned token representation space of the base language model.

Context graphs generated by GCoRe comprise entities and relations between them. Therefore, the model is capable of operating on entity level, whereas Transformer-based models only operate on token level. In the Graph Attention Block of GCoRe, nodes in the context graph undergo updates by gathering information from their neighbors, which are entities that are related to them. As a result, performing graph inference on entity level has positively contributed to the reasoning ability of the model. GCoRe accurately answered a manually crafted question about familial relationships during an interactive experiment, while DeBERTa v3 failed. The context graph produced by GCoRe for the question has illustrated its contributions to the reasoning process.

GCoRe was trained on a single NVIDIA RTX 3090 GPU with 24GB of memory, and its performance was evaluated on two Question Answering tasks: HotpotQA and SQuADv2. GCoRe achieved an improvement of 0.57% in Exact Match and 0.25% in F1 Score on the HotpotQA dataset, which contains long context text and multi-hop questions. On the SQuADv2, a smaller dataset featuring short passages and single-hop questions, GCoRe achieved a marginal improvement of 0.15% in Exact Match and 0.01% in F1 score.

The results of the experiment showed that GCoRe effectively improves the performance of the base language model. It should be noted that the model showed significant improvement on a dataset with long texts, while the improvement on a dataset comprising shorter texts was more moderate. The results suggest that GCoRe captures the global context of longer texts more effectively than the baseline model. Furthermore, utilizing context graphs can enhance performance regardless of whether the key information is spread across multiple passages or concentrated in a single passage.

6.2 Future work

Knowledge graphs are a promising candidate for context graphs in GCoRe because entities in knowledge graphs and their relationships are explicit. It is anticipated that utilizing knowledge graphs as context graphs in GCoRe will yield better outcomes.

The Span Aggregation Block uses a Bi-LSTM layer to combine token representations. To enhance its performance, learned entity embeddings should be implemented to augment external knowledge. Implementing learned entity embeddings could also improve representation alignment and generalization ability. Furthermore, an external knowledge base could enrich the information contained in the context graph.

Instead of adding an extra node representing the question, another entity-level graph for the question could be constructed. There is a potential to improve the model's reasoning and information extraction capabilities through the interaction between entities in the question graph and the context graph. This is a promising direction for future research.

Further analysis of the effectiveness of GCoRe can be achieved by experimenting with other and larger base language models. The concept of utilizing context graphs to enhance token representations in GCoRe can also be applied to other NLP tasks, such as text summarization, document classification, and information retrieval, to improve the processing and comprehension of textual information.

Bibliography

[1] Bahdanau, Dzmitry ; Cho, Kyung H. ; Bengio, Yoshua: Neural Machine Translation by Jointly Learning to Align and Translate. In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2014), 9. – URL https://arxiv.org/abs/1409.0473v7

[2] Battaglia, Peter ; Pascanu, Razvan ; Lai, Matthew ; Rezende, Danilo ; Kavukcuoglu, Koray: Interaction Networks for Learning about Objects, Relations and Physics. In: *Advances in Neural Information Processing Systems* (2016), 12, S. 4509–4517. – URL https://arxiv.org/abs/1612.00222v1. – ISSN 10495258

[3] Battaglia, Peter W. ; Hamrick, Jessica B. ; Bapst, Victor ; Sanchez-Gonzalez, Alvaro ; Zambaldi, Vinicius ; Malinowski, Mateusz ; Tacchetti, Andrea ; Raposo, David ; Santoro, Adam ; Faulkner, Ryan ; Gulcehre, Caglar ; Song, Francis ; Ballard, Andrew ; Gilmer, Justin ; Dahl, George ; Vaswani, Ashish ; Allen, Kelsey ; Nash, Charles ; Langston, Victoria ; Dyer, Chris ; Heess, Nicolas ; Wierstra, Daan ; Kohli, Pushmeet ; Botvinick, Matt ; Vinyals, Oriol ; Li, Yujia ; Pascanu, Razvan: Relational inductive biases, deep learning, and graph networks. (2018), 6. – URL https://arxiv.org/abs/1806.01261v3

[4] Bronstein, Michael M. ; Bruna, Joan ; Lecun, Yann ; Szlam, Arthur ; Vandergheynst, Pierre: Geometric deep learning: going beyond Euclidean data. In: *IEEE Signal Processing Magazine* 34 (2016), 11, S. 18–42. – URL https://arxiv.org/abs/1611.08097v2. – ISSN 10535888

[5] Brown, Tom B. ; Mann, Benjamin ; Ryder, Nick ; Subbiah, Melanie ; Kaplan, Jared ; Dhariwal, Prafulla ; Neelakantan, Arvind ; Shyam, Pranav ; Sastry, Girish ; Askell, Amanda ; Agarwal, Sandhini ; Herbert-Voss, Ariel ; Krueger, Gretchen ; Henighan, Tom ; Child, Rewon ; Ramesh, Aditya ;

Ziegler, Daniel M. ; Wu, Jeffrey ; Winter, Clemens ; Hesse, Christopher ; Chen, Mark ; Sigler, Eric ; Litwin, Mateusz ; Gray, Scott ; Chess, Benjamin ; Clark, Jack ; Berner, Christopher ; McCandlish, Sam ; Radford, Alec ; Sutskever, Ilya ; Amodei, Dario: Language Models are Few-Shot Learners. In: *Advances in Neural Information Processing Systems* 2020-December (2020), 5. – URL https://arxiv.org/abs/2005.14165v4. – ISSN 10495258

[6] Cho, Kyunghyun ; Merriënboer, Bart V. ; Gulcehre, Caglar ; Bahdanau, Dzmitry ; Bougares, Fethi ; Schwenk, Holger ; Bengio, Yoshua: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference* (2014), 6, S. 1724–1734. – URL https://arxiv.org/abs/1406.1078v3. ISBN 9781937284961

[7] Devlin, Jacob ; Chang, Ming W. ; Lee, Kenton ; Toutanova, Kristina: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference* 1 (2018), 10, S. 4171–4186. – URL https://arxiv.org/abs/1810.04805v2. ISBN 9781950737130

[8] Ferrucci, David A.: IBM's Watson/DeepQA. In: *ACM SIGARCH Computer Architecture News* 39 (2011), 6. – ISSN 0163-5964

[9] Geman, Stuart ; Bienenstock, Elie ; Doursat, René: Neural Networks and the Bias/Variance Dilemma. In: *Neural Computation* 4 (1992), 1, S. 1–58. – URL https://www.researchgate.net/publication/220499843_Neural_Networks_and_the_BiasVariance_Dilemma. – ISSN 0899-7667

[10] Gilmer, Justin ; Schoenholz, Samuel S. ; Riley, Patrick F. ; Vinyals, Oriol ; Dahl, George E.: Neural Message Passing for Quantum Chemistry. In: *34th International Conference on Machine Learning, ICML 2017* 3 (2017), 4, S. 2053–2070. – URL https://arxiv.org/abs/1704.01212v2. ISBN 9781510855144

[11] Graves, Alex ; Fernández, Santiago ; Schmidhuber, Jürgen: Bidirectional LSTM networks for improved phoneme classification and recognition. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3697 LNCS (2005), S. 799–804. – URL

https://link.springer.com/chapter/10.1007/11550907_126. – ISBN 3540287558

[12] He, Pengcheng ; Liu, Xiaodong ; Gao, Jianfeng ; Chen, Weizhu: DeBERTa: Decoding-enhanced BERT with Disentangled Attention. In: *ICLR 2021 - 9th International Conference on Learning Representations* (2020), 6. – URL https://arxiv.org/abs/2006.03654v6

[13] Hochreiter, Sepp ; Schmidhuber, Jürgen: Long Short-Term Memory. In: *Neural Computation* 9 (1997), 11, S. 1735–1780. – URL https://www.researchgate.net/publication/13853244_Long_Short-term_Memory. – ISSN 08997667

[14] Honnibal, Matthew ; Montani, Ines ; Van Landeghem, Sofie ; Boyd, Adriane: spaCy: Industrial-strength Natural Language Processing in Python. (2020)

[15] Leviathan, Yaniv ; Matias, Yossi: *Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone*. 2018. – URL https://research.google/pubs/pub49194/

[16] Mikolov, Tomas ; Chen, Kai ; Corrado, Greg ; Dean, Jeffrey: Efficient Estimation of Word Representations in Vector Space. In: *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings* (2013), 1. – URL https://arxiv.org/abs/1301.3781v3

[17] Oono, Kenta ; Suzuki, Taiji: Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. In: *8th International Conference on Learning Representations, ICLR 2020* (2019), 5. – URL https://arxiv.org/abs/1905.10947v5

[18] Ouyang, Long ; Wu, Jeff ; Jiang, Xu ; Almeida, Diogo ; Wainwright, Carroll L. ; Mishkin, Pamela ; Zhang, Chong ; Agarwal, Sandhini ; Slama, Katarina ; Ray, Alex ; Schulman, John ; Hilton, Jacob ; Kelton, Fraser ; Miller, Luke ; Simens, Maddie ; Askell, Amanda ; Welinder, Peter ; Christiano, Paul ; Leike, Jan ; Lowe, Ryan: Training language models to follow instructions with human feedback. (2022), 3. – URL https://arxiv.org/abs/2203.02155v1. ISBN 2203.02155v1

[19] Pennington, Jeffrey ; Socher, Richard ; Manning, Christopher D.: GloVe: Global Vectors for Word Representation. In: *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*

(2014), S. 1532–1543. – URL https://aclanthology.org/D14-1162. ISBN 9781937284961

[20] Peters, Matthew E. ; Neumann, Mark ; Logan, Robert L. ; Schwartz, Roy ; Joshi, Vidur ; Singh, Sameer ; Smith, Noah A.: Knowledge Enhanced Contextual Word Representations. In: *EMNLP-IJCNLP 2019 - 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference* (2019), 9, S. 43–54. – URL https://arxiv.org/abs/1909.04164v2. ISBN 9781950737901

[21] Rajpurkar, Pranav ; Jia, Robin ; Liang, Percy: Know What You Don't Know: Unanswerable Questions for SQuAD. In: *ACL 2018 - 56th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)* 2 (2018), 6, S. 784–789. – URL https://arxiv.org/abs/1806.03822v1. ISBN 9781948087346

[22] Schütt, Kristof T. ; Arbabzadah, Farhad ; Chmiela, Stefan ; Müller, Klaus R. ; Tkatchenko, Alexandre: Quantum-chemical insights from deep tensor neural networks. In: *Nature Communications 2017 8:1* 8 (2017), 1, S. 1–8. – URL https://www.nature.com/articles/ncomms13890. – ISSN 2041-1723

[23] Sutskever, Ilya ; Vinyals, Oriol ; Le, Quoc V.: Sequence to Sequence Learning with Neural Networks. In: *Advances in Neural Information Processing Systems* 4 (2014), 9, S. 3104–3112. – URL https://arxiv.org/abs/1409.3215v3. – ISSN 10495258

[24] Vaswani, Ashish ; Shazeer, Noam ; Parmar, Niki ; Uszkoreit, Jakob ; Jones, Llion ; Gomez, Aidan N. ; Kaiser Łukasz ; Polosukhin, Illia: Attention Is All You Need. In: *Advances in Neural Information Processing Systems* 2017-December (2017), 6, S. 5999–6009. – URL https://arxiv.org/abs/1706.03762v7. – ISBN 1706.03762v7

[25] Veličković, Petar ; Casanova, Arantxa ; Liò, Pietro ; Cucurull, Guillem ; Romero, Adriana ; Bengio, Yoshua: Graph Attention Networks. In: *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings* (2017), 10. – URL https://arxiv.org/abs/1710.10903v3. ISBN 1710.10903v3

[26] Wu, Zonghan ; Pan, Shirui ; Chen, Fengwen ; Long, Guodong ; Zhang, Chengqi ; Yu, Philip S.: A Comprehensive Survey on Graph Neural Networks. In:

IEEE Transactions on Neural Networks and Learning Systems 32 (2019), 1, S. 4–24. – URL http://arxiv.org/abs/1901.00596http://dx.doi.org/10.1109/TNNLS.2020.2978386

[27] Xie, Huiqiang ; Qin, Zhijin ; Li, Geoffrey Y. ; Juang, Biing-Hwang: Deep Learning Enabled Semantic Communication Systems. In: *IEEE Transactions on Signal Processing* 69 (2020), 6, S. 2663–2675. – URL http://arxiv.org/abs/2006.10685http://dx.doi.org/10.1109/TSP.2021.3071210

[28] Xu, Keyulu ; Li, Chengtao ; Tian, Yonglong ; Sonobe, Tomohiro ; Kawarabayashi, Ken I. ; Jegelka, Stefanie: Representation Learning on Graphs with Jumping Knowledge Networks. In: *35th International Conference on Machine Learning, ICML 2018* 12 (2018), 6, S. 8676–8685. – URL https://arxiv.org/abs/1806.03536v2. ISBN 9781510867963

[29] Yang, Zhilin ; Qi, Peng ; Zhang, Saizheng ; Bengio, Yoshua ; Cohen, William W. ; Salakhutdinov, Ruslan ; Manning, Christopher D.: HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original