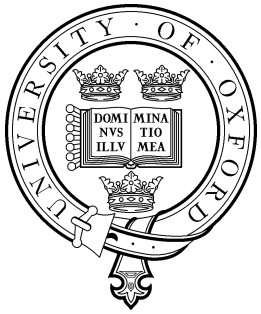
Deep Neural Open Information Extraction with Background Knowledge



Frank Martin Mtumbuka

Linacre College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2022

Acknowledgements

This DPhil research was supported by the Rhodes Scholarship and the Oxford-Linacre African Scholarship. I want to express my sincerest gratitude to the Rhodes Trust, the University of Oxford, and Linacre College for this excellent future. My heartfelt gratitude and appreciation go to my supervisor, Professor Thomas Lukasiewicz, for the opportunity he provided as well as his unwavering belief and support. Nothing would have happened if it had not been for him. My sincere thanks to Professors Phil Blunsom and Nigel Collier for their careful examination of this thesis. I am coming to the end of this phase with a grateful heart and a deep sense of humility and duty. I am grateful to all of my collaborators, especially Patrick Hohenecker, Myeongjun Jang, and Vid Kocijan, for the many stimulating research discussions. I also want to thank colleagues from the department's Intelligent Systems Lab, as well as the department as a whole, for their feedback and remarks on my work. My parents, siblings, and entire family deserve special recognition for their patience, love, and support. To all of my friends who are too many to name and too good to be forgotten, thank you for all of your support.

Abstract

Natural language text, which exists in unstructured format, has a vast amount of knowledge about the world we live in. With the ever-increasing volume of natural language literature, it has become an incredibly time-consuming effort to analyse text and extract important knowledge buried within it. This has resulted in the emergence of information extraction (IE) and natural language processing (NLP) methodologies and tools. IE focuses on the automatic extraction of structured semantic information from text. Extraction and additional analysis of these clear concepts and relationships aid in the discovery of various insights contained in text. This dissertation focuses on open information extraction (OIE), a new type of IE. Unlike traditional IE, OIE is not confined to a predetermined set of domain-specific relations and is expected to extract all relations found in natural language text. Several neural OIE algorithms have been proposed that approach OIE as either a sequence tagging or a sequence generating problem. Sequence tagging approaches identify each token in the input text as belonging to the subject, predicate, or object, whereas sequence generation approaches produce tuples one word at a time given the input text. The proposed methodologies have some limitations, which inspire this research. First, due to unequal label frequencies in OIE datasets, sequence tagging techniques tend to put too much emphasis on labels that appear at higher frequencies. Second, sequence generation systems are prone to not only producing the same fact several times, but also

producing repeating tokens in facts. Third, while sequence generation systems use words from a vocabulary while constructing implicit facts, they lack features that explicitly encourage them to either use terms from the vocabulary or input text. Fourth, techniques that incorporate syntactic information by employing part-of-speech (PoS) and dependency tags in addition to the actual input text make insufficient use of the vast syntactic information, particularly that reflected in the structure of dependency trees. This dissertation aims to address the aforementioned drawbacks of earlier OIE approaches. In this research, I only examine neural OIE approaches because they have outperformed prior rule-based systems and address the issue of error propagation in rule-based systems. I give novel methodologies to tackle the limits of earlier approaches while leveraging state-of-the-art (SOTA) deep learning methods. Furthermore, I study whether incorporating factual knowledge from knowledge graphs (KGs) into neural OIE models can improve the performance of OIE approaches. First, I present three innovative training procedures for sequence labelling OIE approaches to remove model biases caused by uneven tag frequencies in OIE datasets on models. Second, I avoid creating redundant tokens in facts and increase the models' capacity to create implicit facts by providing methods for explicitly guiding the models to use terms from either the vocabulary or input text. This strategy drastically reduces the number of recurring tokens in fact. Furthermore, when this strategy is used, the models repeat fewer tokens from the original phrase and introduce more tokens from the vocabulary, which implies a better ability to generate implicit facts. Third, I suggest a method for maximising the syntactic information provided by dependency tree topologies. Using the structure of dependency trees, I compute syntactically rich vector repre-

sentations of input text tokens. Fourth, I present a knowledge-enhanced OIE framework for both sequence tagging and sequence generation OIE techniques, which builds on recent achievements in embedding knowledge in pre-trained language models (PLMs). After thorough testing, I confirm that the knowledge-enhanced OIE framework improves the performance of OIE models. Finally, I provide a unique discriminative strategy for neural OIE model training.

Contents

List of Figures

List of Tables

List of Acronyms

AUC-PR area under the precision-recall curve

BERT bidirectional encoder representations from transformers

BiLSTM bidirectional LSTM

CNN convolutional neural network

CRF conditional random field

DL deep learning

GAN generative adversarial net

GAT graph attention network

GNN graph neural network

GRU gated recurrent unit

IE information extraction

KB knowledge base

KE knowledge embedding

KG knowledge graph

LSTM long short-term memory

MLM masked language modelling

MLP multi-layer perceptron

NLL negative log-likelihood

NLP natural language processing

NLU natural language understanding

NMT neural machine-translation

NN neural network

OIE open information extraction

OOV out-of-vocabulary

PGN pointer-generator-network

PLM pre-trained language model

PoS part-of-speech

PSL probabilistic soft logic

RNN recurrent neural network

SOTA state-of-the-art

SRL semantic role labelling

Chapter 1

Introduction

In this chapter, I describe and motivate research gaps that this dissertation addresses. I present the research questions guiding this research. Finally, I present the contributions of this research and the structure of the dissertation.

1.1 Thesis Motivation

A tremendous wealth of knowledge about the world that we live in is encoded in natural language text, which exists in unstructured format. With the ever-increasing volume of natural language text, it has become an impossibly tedious task to analyze text and gain valuable knowledge embedded in it. This has led to the rise of IE and NLP techniques and tools. IE from text focuses on the automatic extraction of structured semantic information from text [Jurafsky and Martin, 2009, Niklaus et al., 2018]. The extraction and further analysis of these explicit concepts and relationships help in discovering multiple insights contained in text.

Traditional IE techniques are concerned with discovering a predefined set of relations in a small and homogeneous corpus [Niklaus et al., 2018]. This is accomplished by combining the target relations with handcrafted extraction patterns or patterns learned from hand-labelled homogeneous training instances as input. This means

that in order to extract information in a new domain, the user must specify a new set of relations in that domain as well as construct a new set of extraction rules or training examples. As a result, classic IE approaches rely on extensive manual effort and are not scalable to heterogeneous corpora, making them ineffective in domain-independent scenarios. Banko et al. [2007] presented a new IE paradigm, OIE, in response to the limitations of classic IE techniques. OIE is not restricted to a certain set of domain-specific relations and is expected to extract all relations found in natural language text. OIE enables domain-independent discovery of relations from text and scales across huge, heterogeneous corpora such as the Web.

The OIE relational tuples are presented in a structured representation that consists of a set of arguments and a relational phrase. The relational phrase expresses the semantic relationship between the arguments. The predicate is another name for the relational phrase. In most cases, the subject and object in a given piece of text are the arguments. For instance, given the sentence »*Sam succeeded in convincing John.*«, an OIE system should return the tuple ⟨*Sam*, *succeeded in convincing*, *John*⟩. The relation phrase »*succeeded in convincing*« indicates the semantic relationship between the subject »*Sam*« and the object »*John*«. A tuple is also commonly referred to as a fact in OIE. In this work, I use the words tuple and fact interchangeably. OIE plays a key role in NLP and natural language understanding (NLU) , and thus fosters many downstream NLP and NLU applications, such as knowledge base (KB) construction from text [Soderland et al., 2010], question answering [Fader et al., 2014], information retrieval [Etzioni, 2011], and text comprehension and entailment [Mausam, 2016].

TextRunner [Banko et al., 2007] stands out as the first OIE system. After TextRunner, a number of OIE systems have been developed, such as ClauseIE [Corro and Gemulla, 2013], ProPS [Stanovsky et al., 2016], and OpenIE4 [Angeli et al., 2015]. These systems are mostly rule-based and use the language syntax to extract triples from sentences. For example, ClauseIE makes use of linguistic knowledge of

the grammar of a sentence to detect clauses, and to identify the type of any such. OpenIE4 uses semantic role labeling to extract tuples from a sentence, and PropS relies on the proposition structure of syntax and dependency trees of the input sentence. However, rule-based systems suffer from severe error propagation when applied to examples outside of expected language patterns [Cui et al., 2018].

Several neural OIE methods have been developed that approach OIE as either a sequence tagging [Stanovsky et al., 2018, Roy et al., 2019, Jiang et al., 2019, Zhan and Zhao, 2020] or a sequence generation problem [Cui et al., 2018, Sun et al., 2018, Kolluru et al., 2020]. Sequence tagging approaches label each token in the input text as belonging to the subject, predicate, or object, whereas sequence generation approaches produce tuples one word at a time given the input text.

Sequence tagging OIE methods use BIO tags [Ratinov and Roth, 2009] to label each token in the input text. Each element of a tuple is implicitly extracted by labeling the according sequence of words with a **B**eginning-tag followed by an arbitrary number of **I**nside-tags, and words that are not part of any extracted phrase are marked with the **O**utside-tag. Tags are prefixed with A0 if they refer to a tuple's subject, P for the predicate, and A1 to refer to the object of the extracted triple. Figure 1.1 illustrates how BIO tags are used to extract tuples from text. The *O*-tag, which indicates tokens that are not part of the subject, predicate, or object, appears far more frequently than the other tags among all BIO tags. As a result, the ability to properly predict the *O*-tags has the greatest direct influence on the negative log-likelihood (NLL) and encourages models to place too much emphasis on this label. Furthermore, sequence tagging methods can only tag tokens that are explicitly present in the input text. As a result, these techniques are unable to extract implicit facts from text.

OIE techniques based on a sequence generation approach produce facts one word at a time based on the input text. As such, they can incorporate words from a vocabulary or change the order of words when generating facts from an input text [Kolluru

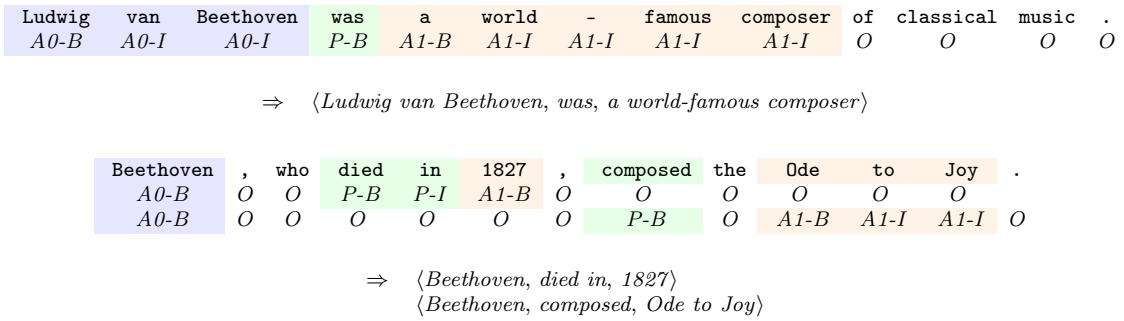


Figure 1.1: Examples of tuple extraction using BIO tags in OIE.

et al., 2020]. This property enables these approaches to yield tuples that are implicitly present in the input text, because implicit relations require some words that are not explicitly included in the input text. Despite being superior to sequence tagging approaches, sequence generation approaches are prone to not just producing the same fact numerous times, but also producing repeating tokens in facts. This has an impact on the quality of information being extracted by these systems.

Furthermore, while generating facts, sequence generation systems lack procedures that explicitly direct them to "select" the next word from either the vocabulary or input text. This limits the ability of such approaches to generate implicit facts. Moreover, several OIE approaches incorporate syntactic information by using PoS and dependency tags in addition to the actual input text [Stanovsky et al., 2018, Zhan and Zhao, 2020]. When these tags are utilised in a piece of work, their embeddings are concatenated with the embeddings of the associated text tokens. This formulation makes insufficient use of the extensive syntactic information, particularly that expressed in the structure of dependency trees.

The research in this dissertation is motivated by the aforementioned limitations of prior OIE methodologies. In this study, I am only looking into neural OIE techniques because they have outperformed previous rule-based systems and address the problem of error propagation in rule-based systems [Cui et al., 2018]. I present novel methodologies to tackle the limitations of earlier neural approaches while utilising SOTA deep

learning methods. Furthermore, I study whether incorporating factual knowledge from KGs into neural OIE models can increase OIE models' performance. Finally, I provide a novel discriminative training strategy for neural OIE models.

1.2 Research Questions

In order to address the limitations of neural OIE approaches highlighted above, I formulate research questions to help guide this research. As mentioned before, this work only focuses on OIE approaches that are based on neural networks (NNs) [Stanovsky et al., 2018, Cui et al., 2018, Jiang et al., 2019, Zhan and Zhao, 2020, Sun et al., 2018, Kolluru et al., 2020] because they outperform previous rule-based systems and already address some rule-based system issues like error propagation. The work presented in this dissertation is guided by the following research questions:

- RQ1: Some tags appear at considerably higher frequency than others in OIE sequence tagging models. As a result, the models tend to focus too much on tags that emerge at higher frequencies, impairing their ability to correctly predict tags that appear at lower frequencies. For instance, Stanovsky et al. [2018] point out that O-tags make up 58 percent of the OIE2016 dataset, and that their model, which was trained on it, predicts shorter arguments. This means the model learns to predict additional O-tags, the majority of which are incorrectly predicted as part of the tuple arguments. **What approaches can be put in place to ensure that the models are still able to predict tags that appear in lower frequencies correctly?**

  The effectiveness of the proposed approaches to lessen the effects of tag imbalance will also be tested in terms of how well the model predicts other labels, in addition to the conventional F_1 and area under the precision-recall curve (AUC-PR) metrics for evaluating OIE models. More precisely, I'll look

at the percentage of O-tags that are successfully predicted against those that are mistakenly predicted as part of either the subject, predicate, or object when using the approaches developed in this work. If the approaches lower the fraction of O-tags predicted as tuple arguments, they are effective in mitigating the influence of tag frequency imbalance on the model's performance.

- RQ2: Sequence generation approaches are susceptible to generating facts that often express redundant information and are also prone to generating repetitive text in facts. Sun et al. [2018] and Kolluru et al. [2020] investigate the problem of repeatedly generating the same facts, but not the problem of generating repeating text in facts, which is likewise redundant information. **How can I build on this work to make models less prone to generating repetitive text in facts?**

  In addition to the usual $F_1$and AUC-PR metrics, the success of the suggested ways to decrease repetitive tokens in generated tuples will be assessed by comparing the percentage of repetitive tokens before and after the proposed approaches are applied. A decrease in the average proportion of repetitive tokens after using the proposed techniques would indicate that the techniques are effective in addressing the problem of generating repetitive tokens in tuples.

- RQ3: Sequence generation approaches are capable of introducing words that are not in the input sentence into a generated fact. This capability enables such approaches to generate facts that are implicitly stated in the input sentence. However, the approaches employed so far by Cui et al. [2018] and Sun et al. [2018] are restrictive in how the models "pick" words when generating facts which restrains the models' flexibility in generating implicit facts. **How can I enhace the models' flexibility in generating implicit facts?**

  Implicit facts are those that are not explicitly expressed in the input text. The

models must incorporate new words into the facts during fact generation to indicate what is implied in the input text in order to generate facts. The model's capacity to add new words into the fact would be measured intuitively by looking at the intersection of the words in the input text and the generated fact. If the number of words in the intersection decreases after using the proposed strategies, it means the model is copying fewer words from the input text and introducing more words from the vocabulary into the fact. As a result, the model is better equipped to predict implicit facts.

- RQ4: Some OIE methods make use of PoS and dependency tags in addition to the actual input text as a way of incorporating syntactic information [Stanovsky et al., 2018, Zhan and Zhao, 2020]. In such work where these tags are used, the embeddings for the tags are just concatenated to the embeddings of the corresponding text tokens. This formulation does not fully use the rich syntactic information, especially the one that is expressed in the structure of dependency trees. **How can I make best use of syntactic features to improve the performance of OIE models?**

- RQ5: Recently, many pieces of work have investigated how to incorporate knowledge into PLMs [Baldini Soares et al., 2019, Zhang et al., 2019, Peters et al., 2019, Wang et al., 2021] and have shown that this achieves significant improvements on various knowledge-driven tasks. **Can neural OIE models benefit from factual knowledge present in KGs?**

1.3 Contributions

This work makes significant contributions to the field of OIE by investigating the research questions indicated in Section 1.2. In Chapters 3, 4, and 5, I go over the contributions in great depth. The main contributions of this dissertation are briefly

summarized as follows:

1. **New training schemes for OIE as a sequence tagging problem (Chapter 3).**

   I develop three unique training procedures for OIE as a sequence labelling problem to remove the model bias that is caused by unequal tag frequencies in OIE datasets. To summarise, the first training technique attempts to limit the influence of O-tags on the loss term by completely ignoring them. The second training approach disregards all probabilities except those of positions that exist soon before or quickly after a change. Only the initial and last places of each extracted triple element are optimised in the third training scheme. These unique training schemes surpass the natural NLL formulation, which takes into account all points in the sequence when computing the loss. The developed training schemes can be used for any work that qualifies as a sequence tagging problem.

2. **A new method of computing syntactically rich vector representations of input tokens guided by the structure of dependency trees (Chapter 4).**

   I look into ways to make better use of the rich syntactic information that is available when computing embeddings, notably that which is contained in the structure of dependency trees. To that end, I suggest a method for maximising the use of the syntactic information that is provided by the structure of the dependency tree. Using the structure of dependency trees, the proposed method computes syntactically rich vector representations of input text tokens. Based on the structure of the dependency tree, I generate a visibility matrix of the tokens in each input sentence. Tokens in the dependency tree that are directly related to each other are considered visible to each other. This visibility matrix

is used to compute vector representations in such a way that only visible tokens are taken into account. This method generates embeddings that improve the performance of OIE models.

3. **A novel strategy for training neural OIE models (Chapter 4).**

   I present a novel training technique for neural OIE models. I introduce a new module, the discriminator, which takes the produced fact as input and classifies its tokens as "real" or "fake", with "real" tokens being those that are in both the generated and gold facts and "fake" tokens being those that are in the generated fact but not in the gold fact. Based on experimental findings, I conclude that using discriminative training significantly improves the performance of OIE models.

4. **Methods for eliminating redundant words in generated facts and enhancing the ability of OIE models to generate implicit facts (Chapter 4).**

   I concentrate on sequence-generating algorithms in particular, since they are prone to producing facts that frequently convey redundant information. Previous pieces of work consider redundant information to be only the generation of the same fact multiple times. They do not look into the issue of repeating words in the generated facts. In addition, their methodologies are restrictive in how the models "select" words when generating facts, limiting the models' flexibility in generating implicit facts. I address this by providing methods for explicitly guiding models to either "select" terms from a vocabulary or the input text in order to control the formation of repeated text in facts. This strategy reduces the number of recurring tokens in facts significantly. Furthermore, when applying this method, the models replicate fewer tokens from the source sentence and introduce more tokens from the vocabulary, resulting in an improved ability to

produce implicit facts.

5. **Paraphrased versions of the CaRB, IOE2016, and LSOIE datasets (Chapter 4).**

   To further analyse and improve the ability of OIE models to generate implicit facts, I created paraphrased versions of the CaRB Bhardwaj et al. [2019], IOE2016 Stanovsky and Dagan [2016], and LSOIE Solawetz and Larson [2021] datasets. I only paraphrase the training sets of the datasets and keep the test sets. The meaning of the paraphrased samples is the same as the meaning of the original samples, but the lexical form is different. Models trained on this data do better at capturing implicit relationships. Additionally, when trained on augmented datasets, models improve in performance.

6. **A framework for incorporating factual knowledge housed in KGs into neural OIE models (Chapter 5).**

   I investigate how KG prior information can be used to improve the performance of neural OIE algorithms. In order to achieve this goal, I present a knowledge-enhanced OIE framework for both sequence tagging and sequence generation OIE approaches, which builds on recent breakthroughs in embedding information in PLMs. After thorough testing, I observed that the knowledge-enhanced OIE architecture improves the performance of OIE models.

1.4 Thesis Structure

In Chapter 2, I examine the current SOTA neural OIE techniques. I go over these approaches in depth and highlight the gaps that exist in the present approaches that this study addresses. I also discuss previous attempts to incorporate knowledge into language models.

In Chapter 3, OIE is viewed as a sequence tagging task. In this chapter, I propose novel training schemes for sequence tagging approaches that reduce the impact of skewed tag frequencies in datasets on the models' ability to predict correct tags. I outline the approach and provide detailed descriptions of the experiments that I ran to demonstrate the efficacy of the recommended training schemes. Finally, I present a thorough examination of the study results and ablations.

In Chapter 4, OIE is viewed as a sequence generation task. I offer various new strategies for training neural OIE models in this chapter. To begin, I offer a novel method for calculating syntactically rich text embeddings that is directed by the structure of dependency trees. Second, I propose a new module to classify tokens in the generated fact as "real" or "fake", corresponding to tokens that are in both the generated and gold fact or those that are in the generated fact but not in the gold fact, respectively. In addition, I address the issue of repetitive tokens in generated facts and increase the models' capacity to generate implicit facts.

In Chapter 5, I take advantage of recent advancements that attempt to incorporate knowledge from KGs into PLMs and create a framework for integrating real-world knowledge into OIE models. I show that having significant background knowledge helps OIE models to perform better.

In Chapter 6, I provide a summary of the important findings reported in this dissertation, as well as how the contributions produced by this research address the research questions posed in Section 1.2. In addition, I outline possible future avenues for this work.

1.5 Publications from this Work

Table 1.1 summarises three manuscripts that have resulted from the research that has been conducted for this dissertation. The first manuscript, (1), was accepted for publication at a peer-reviewed conference. The second manuscript, (2), is currently being peer-reviewed, and the third, arising from Chapter 5, is yet to be submitted for review. In addition, I have worked with others on projects that are not related to this dissertation.

	Publications	**Related Chapter**
(1)	Patrick Hohenecker[1], Frank Mtumbuka[1] , Vid Kocijan, and Thomas Lukasiewicz. Systematic comparison of neural architectures and training approaches for open information extraction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 8554–8565. Association for Computational Linguistics.	3, 5
(2)	Frank Mtumbuka, and Thomas Lukasiewicz. Syntactically rich discriminative training: A powerful approach for open information extraction. *Under review.*	4, 5
	Other Publications	
(3)	Myeongjun Jang, Frank Mtumbuka, and Thomas Lukasiewicz. Beyond distributional hypothesis: Let language models learn meaning-text correspondence. In *Findings of NAACL 2022, Seattle, Washington, USA, July 2022. Association for Computational Linguistics.*	

Table 1.1: Papers and manuscripts from this dissertation's and related research.

[1]Equal contribution.

Chapter 2

Literature Review

This chapter provides an overview of current SOTA neural OIE techniques. There are two types of neural OIE approaches: sequence tagging approaches and sequence generation approaches. I go over both of these approaches in detail and point out the shortcomings in the present methodologies that this study aims to remedy. I also discuss previous attempts to incorporate knowledge from KGs into PLMs. The audience for this dissertation is assumed to have a basic understanding of deep learning and NLP.

2.1 Sequence Tagging Methods

Sequence tagging OIE approaches employ BIO tags to identify each token in the input text as belonging either to the subject, predicate, or object. As demonstrated in Figure 1.1, each element of a tuple is implicitly retrieved by labelling the associated sequence of words with appropriate tags. It is important to note that systems in this category can only label tokens that appear in the input sequence and in the order specified in the input sequence. As a result, they are unable to reorder tokens in extractions or introduce new tokens to accommodate what is implicitly expressed in the input sequence. In this section, I describe and discuss numerous neural models

that approach OIE in this manner.

Stanovsky et al. [2018] created a neural OIE model for binary extractions based on sequence tagging. A binary extraction occurs when an input sentence contains only one predicate span, and all other spans are either subject or object spans. They create an appropriate sequence of PoS tags for each word sequence in the input sentence. In addition, for a given sentence, they extract the index of the syntactic head of the predicate. Each word's feature vector is created by concatenating the word's embedding, the embedding of its associated PoS tag, and the embedding of the predicate. They duplicate the predicate head's features on all words so that the model can access this information more directly when making predicate-specific word label predictions. The generated feature vectors are then passed into a bidirectional LSTM (BiLSTM), which computes contextualised feature vectors. A Softmax function is then applied to these contextualised feature vectors to generate probability distributions over probable BIO tags. The confidence of an extraction is calculated by multiplying the probabilities of the **B** and **I** tags that are involved in the extraction.

During training, Stanovsky et al. [2018] create a set of tuples of a given sentence by grouping them using the predicate syntactic head word, allowing them to run the model once for each predicate. Though this makes learning predicate specific tuples from a given sentence easier for the model, this order of operation is lacking in the real world. Furthermore, they report that their model predicts shorter tuple arguments, implying that the model predicts more tags indicating tokens that are not part of the subject, predicate, or object. This is due to the fact that such words, those labelled with the O-tag, appear in the dataset at a much higher frequency (58%) than the rest of the words [Stanovsky et al., 2018].

Jiang et al. [2019] conduct confidence modelling, building on the work of Stanovsky et al. [2018]. They are working to improve the precision and recall of extracted facts by ranking extractions based on their estimated quality. This is significant

for downstream applications that rely solely on the tradeoffs between the extracted facts' precision and recall. The confidence score of the extracted tuple is computed in most current OIE systems by looking at the extraction likelihood rather than the extraction quality [Stanovsky et al., 2018]. This ignores a phenomenon in which incorrect extractions of one sentence may have higher confidence scores than correct extractions of another sentence. This is due, in part, to a misalignment between training and test-time objectives. The systems are only trained to maximise the likelihood of gold extraction during training. To address this, Jiang et al. [2019] train an additional module to classify the extracted tuples as correct or incorrect. The binary classification loss in this step is used to increase confidence in correct extractions while decreasing confidence in incorrect ones. The correct extractions are then added to the training data, and the model's performance improves. The proposed method has the advantage of not requiring any changes to the base model, and the binary classification loss only serves as additional supervision. After one round of training, they note that the resulting model improves not only at confidence modelling but also at assertion generation, implying that higher-quality extractions can be used as training samples to iteratively continue this training process.

Zhan and Zhao [2020] present a span OIE model for n-ary tuple extraction that uses the sequence tagging approach. A span of words in the input phrase is labelled by the model as belonging to a predicate span or an argument span. The model is made up of the predicate module and the argument module. The predicate module looks for predicate spans in a given sentence, whereas the argument module looks for argument spans given a predicate span and a sentence as its input. The feature vector of each word in a given sentence is produced by concatenating the word embedding, the PoS tag embedding, the dependency tag embedding, and the embedding of a binary value indicating whether the word is part of the predicate span or not. These feature vectors are then fed into a BiLSTM network, which computes contextualised

feature vectors. The contextualised feature vectors are then sent into a feedforward network, followed by a Softmax layer, to acquire the scores of various tags for each span.

Zhan and Zhao [2020] are cautious in their approach because the number of spans to be considered per sentence grows exponentially as the sentence length increases. This will inevitably reduce the performance of their model and increase the computational cost. As a result, they limit the number of spans by limiting their length, ensuring that the spans do not overlap with the predicate span, and requiring that each span have a syntactic head. They observe improved model performance with these span constraints. They also experimented with whether or not syntactic information from dependency trees should be included in their model. When they include syntactic information, they concatenate the dependency tag embedding to the corresponding word, and they observe that such models produce lower results than when dependency tag embedding is not used. They attribute this to noise in the training corpus and ineffective dependency tree generation tools. However, I believe that their poor performance is due to the way they incorporate syntactic information. Surprisingly, the syntax-aware models performed better in some cases.

On benchmark datasets, sequence tagging techniques for OIE have produced good results. However, they contain gaps that require additional research in order to increase their effectiveness. For starters, the presented approaches do not address the issue of uneven BIO tag frequencies in datasets. The *O*-tag, which denotes tokens that are not part of a subject, predicate, or object, appears far more frequently than other BIO tags. As a result, the models give more focus to this label than to other labels that exist at lower frequencies. This has an immediate impact on the quality of OIE tuples produced by systems trained on datasets with such a tag frequency imbalance. Stanovsky et al. [2018] observe that the *O*-tags account for 58% of the labels in the OIE2016 [Stanovsky and Dagan, 2016] dataset, and their model,

which was trained on this dataset, predicted shorter tuple arguments. I believe that the models predicted a greater number of O-tags as part of the tuple arguments. There has been no research into reducing the impact of skewed tag frequencies in the dataset on the extraction quality. Second, Stanovsky et al. [2018] and Zhan and Zhao [2020] incorporate syntactic information by using PoS and dependence tags in addition to the actual input text. When these tags are utilised in a piece of work, the embeddings for the tags are simply concatenated with the embeddings of the associated text tokens. Despite the fact that syntactic information improves extraction quality, Zhan and Zhao [2020] find that when dependency tag embeddings are concatenated to the corresponding word, their model produces lower results than when dependency tag embeddings are not used. I hypothesise that this formulation, in which dependency tag embeddings are simply concatenated to the corresponding words, makes insufficient use of the extensive syntactic information, particularly that expressed in dependency tree topology. Third, sequence tagging approaches only tag terms that appear explicitly in the input sentence. They are unable to capture what is implied in a specific piece of writing. This feature naturally prevents these systems from generating tuples that are implicitly conveyed in text, preventing them from being used in practice, particularly in situations where most knowledge is expressed implicitly.

2.2 Sequence-to-Sequence Methods

Sequence generation OIE techniques generate tuples one word at a time from an input sentence. Approaches in this category are superior to sequence tagging approaches in that they can introduce words from the entire vocabulary rather than just the words in the input sentence and can also change the order of words when generating tuples. Kolluru et al. [2020], for example, observe that sequence generation approaches are

particularly capable of introducing auxiliary words in tuples. They can generate a tuple ⟨*Trump*, *is president of*, *US*⟩ given a sentence »*US President Trump ...*«. Because sequence generation approaches can change word order and insert new words into tuples, they are better suited to generating tuples that are implicitly specified in the input sentence. This section discusses recent neural approaches that have been proposed in this category. I also highlight their drawbacks.

Cui et al. [2018] create a neural OIE model that generates binary extractions from a given text. Their model, CopyAttention, generates a tuple one word at a time while being conditioned on the input sentence and the previously generated token of the tuple. They make a strong assumption in their approach that both the predicate and argument spans are strictly sub-spans of the input sentence. In both the encoder and the decoder, CopyAttention employs stacked layers of LSTMs. The encoder takes the input sequence and encodes it into a set of hidden states. Using the attention technique proposed by Bahdanau et al. [2015], the context vector is calculated from the encoder's hidden states. The context vector, the decoder's hidden state, and the previously predicted token of the tuple are supplied into the decoder, which outputs a feature vector that is fed into a Softmax function to compute the probability distribution over the vocabulary. Copying methods [See et al., 2017] are employed to ensure that CopyAttention only outputs tokens from the input sentence. As a result, the model only picks words that are present in the input sentence.

The copying mechanism as applied by Cui et al. [2018] is especially significant in the neural OIE task because the output vocabulary is derived directly from the input vocabulary, with the exception of placeholder symbols. This forces the model to only use words from the input vocabulary. On the other hand, limiting the vocabulary to simply the input vocabulary and placeholder symbols has a direct impact on the model's capacity to introduce new terms in generated tuples and hence on the model's ability to construct implicit facts.

Sun et al. [2018] present Logician, a sequence-to-sequence approach that maps a given sentence to a set of facts. Logician employs three ways to accomplish this task: restricted copy, coverage, and gated dependence attention. The restricted copy technique restricts the model to only "select" words from the input sentence or a list of preset keywords that represent a subset of the full vocabulary. This allows the model to try to use the words that are in the original sentences to express tuples as much as feasible. The coverage mechanism controls under-extraction and over-extraction. Under-extraction occurs when the model extracts only some tuples and ignores others, whereas over-extraction occurs when the model extracts the same fact many times. To control under-extraction and over-extraction, Logician employs a coverage vector, which is the sum of all attention scores over the words in the input phrase up to timestep t. The gated dependency attention is employed during decoding to ensure that the candidate words and the previously decoded words have a meaningful semantic relationship. To build feature vectors of text tokens, Logician leverages bidirectional gated recurrent units (GRUs).

Kolluru et al. [2020] investigate the problem of over-extraction and propose IMoJIE, a new sequence-to-sequence model. IMoJIE generates tuples using sequential decoding while conditioned on the input text and previously decoded tuples. A decoded tuple is attached to the input sentence during each decoding cycle. This creates a new input into the model for the following iteration. The model iterates until an *EndOfExtractions* token is generated. IMoJIE employs a bidirectional encoder representations from transformers (BERT) encoder and an LSTM decoder.

Despite the fact that sequence-to-sequence methods for OIE are superior to sequence tagging approaches, the sequence-to-sequence approaches that have been developed have several constraints that hinder their superiority from being manifested. When creating a tuple, Logician, for example, employs a restricted copy technique that allows the model to just copy words from the input sentence or a set of se-

lected key terms from the lexicon. Furthermore, CopyAttetion assumes that both the predicate and argument spans are strictly sub-spans of the input sentence. This prevents the model from considering all of the terms in the vocabulary. As a result, the model's ability to construct implicit tuples is limited, as the models must employ new terms from the full vocabulary to represent tuples that are implicitly stated in the input sentences. Furthermore, Logician and IMoJIE make an attempt to solve under-extraction and over-extraction difficulties in OIE. To overcome the aforementioned difficulties, Logician employs coverage mechanisms, whereas IMoJIE employs sequential decoding mechanisms. Both pieces of work are concerned with reducing the problem of redundant information that arises as a result of extracting the same tuple many times. They are unconcerned about repetitive tokens in tuples that occur from repeatedly duplicating the same word from either the vocabulary or the input text, which affects the tuple quality. This issue is also unaddressed by CopyAttention.

2.3 Injecting Knowledge into Language Models

Many studies have recently been conducted to examine ways to incorporate world knowledge into PLMs, and they have proven to improve the knowledge content of PLMs, particularly factual knowledge.

Baldini Soares et al. [2019] use a simple "matching the blanks" pretraining objective to assist in the classification of relationships. To begin, they generate a training dataset of relation statements in which entities are substituted with a special "blank" symbol. During training, they select a pair of blank-containing relations that share the same entities and apply a training objective that encourages the relation representations to be similar, because the relations share the same entities. Their model learns fixed vectors for relation representation from entity-linked relation statements alone as part of the process. Their findings show that models created with these

representations outperform previous work on relation classification by a significant margin, even with a limited number of training samples.

Zhang et al. [2019] create ERNIE, which trains a language representation model using textual corpora as well as KGs. They detect entity mentions in text and match them to entities in the full KG. KE algorithms are utilised to generate the embeddings of the respective KG entities, which are subsequently fed into ERNIE. KE algorithms are algorithms that learn a low-dimensional representation of KG's entities and relations while preserving their semantic meaning. ERNIE integrates entity representations in the knowledge module into the underlying layers of the semantic module based on alignments between text and KGs. They mask off part of the named entity alignments in the input text during training and prompt the model to choose appropriate entities from KGs to complete the alignments. As a result, the model learns to combine context and knowledge facts in order to predict tokens and entities. This yields a well-informed language representation model. More specifically, they discover that their approach improves performance on knowledge-driven tasks such as entity typing and relation classification.

Peters et al. [2019] propose a general method for embedding multiple KBs into large-scale models, KnowBert, and thus enriching their representations with structured, human-curated knowledge. They use an integrated entity linker to retrieve relevant entity embeddings for each KB, then update contextual word representations using a type of word-to-entity attention. Unlike previous approaches, the entity linkers and self-supervised language modelling objectives are jointly trained end-to-end in a multitask setting that combines a small amount of entity linking supervision with a large amount of raw text. The key concept is to explicitly model entity spans in the input text and then use an entity linker to retrieve relevant entity embeddings from a knowledge base to form knowledge-enhanced entity-span representations. The model then re-contextualizes the entity-span representations using word-to-entity at-

tention to enable long-distance interactions between contextual word representations and all entity spans in the context. After incorporating WordNet and a subset of Wikipedia into BERT, KnowBert exhibits improved perplexity, ability to recall facts as measured in a probing task, and improved performance on relationship extraction, entity typing, and word sense disambiguation.

Logan et al. [2019] observe that language models are only capable of remembering facts that they have seen during training, and these facts are frequently difficult to recollect. To solve this, they create a knowledge graph language model that includes methods for detecting and copying facts from a knowledge graph that are relevant to the given context. This method enables the model to render previously unseen information as well as construct out-of-vocabulary tokens. Hayashi et al. [2020] present a latent relation language model that parametrizes the joint distribution across words in a document and entities that occur in them via knowledge graph relations. This method enhances the models' ability to learn to predict relevant relationships in a given context. Both [Logan et al., 2019] and [Hayashi et al., 2020] train superior generation models using relations between entities within a single sentence.

Wang et al. [2021] emphasise that while PLMs can learn useful linguistic knowledge from unlabeled text, they cannot capture factual knowledge. On the other hand, KE methods effectively represent relational facts in KGs with informative entity embeddings, but conventional KE models cannot fully exploit the abundant textual information. To that end, Wang et al. [2021] propose KEPLER, a unified model for knowledge embedding and pre-trained language representation that integrates factual knowledge into PLMs and computes effective text-enhanced knowledge embedding with strong PLMs. Their method encodes textual entity descriptions as entity embeddings using PLMs and then optimises the knowledge embedding and language modelling objectives jointly. They use entity descriptions to bridge the KE-PLM gap and align the semantic space of text with the symbol space of KGs. They confirm,

through extensive experiments, that KEPLER is capable of integrating factual knowledge into language representation under the supervision of KGs and the KE objective. Moreover, by the MLM objective, KEPLER inherits the strong ability of language understanding from PLMs. Furthermore, the KE objective improves KEPLER's ability to extract knowledge from text by requiring the model to encode entities from their corresponding descriptions.

Previous attempts to incorporate factual knowledge into PLMs have proven to improve PLMs' performance on a variety of NLP tasks, particularly knowledge-driven tasks like entity typing and relation classification [Baldini Soares et al., 2019, Zhang et al., 2019, Peters et al., 2019, Wang et al., 2021]. In the case of KEPLER, this is due to the fact that incorporating knowledge into PLMs improves their ability to extract knowledge from text by requiring the model to encode entities from their corresponding descriptions. This property could also be very useful to OIE, as OIE focuses on extracting knowledge from natural text in the form of triples. In this research, I extend KEPLER, which computes entity representations based on textual descriptions, allowing OIE models to develop more context-aware and knowledgeable embeddings, and propose a knowledge-enhanced framework for OIE.

Chapter 3

Systematic Comparison of Neural Architectures and Training Approaches for Open Information Extraction

In this chapter, I will look at OIE as a sequence tagging problem. I propose novel training techniques for sequence tagging systems that lessen the impact of skewed tag frequencies in datasets on the models' ability to predict correct tags. These training approaches are applicable to any domain in which tasks may be expressed as sequence tagging problems. I outline the approach and provide detailed descriptions of the experiments that were ran to demonstrate the effectiveness of the recommended training schemes. Finally, I give a thorough analysis of the findings and ablations.

3.1 Introduction

The field of IE focuses on the automatic acquisition of structured information from text, and approaches from this field are used to extract needed information from enormous corpora of text. Unlike traditional IE, OIE predicates and entities are not domain-specific and are not predefined. In more formal terms, an OIE algorithm seeks to extract all facts implied by the input text. The extracted facts are in the form of $\langle$*subject*, *predicate*, *object*$\rangle$ triples. Alternative formulations allow for longer tuples, but most pieces of work including what is presented in this chapter focus solely on binary predicates. Given the text »*Sam was successful in convincing John.*«, an OIE system should provide the following tuple: $\langle$*Sam*, *was successful in convincing*, *John*$\rangle$. The relation phrase »*was successful in convincing*« denotes the semantic connection between »*Sam*« and »*John*«.

Several sequence tagging OIE techniques [Stanovsky et al., 2018, Zhan and Zhao, 2020, Cui et al., 2018] have produced promising results on benchmark datasets. Sequence tagging OIE approaches employ BIO tags to identify each token in the input text as belonging either to the subject, predicate, or object. SpanOIE, developed by Zhan and Zhao [2020], formalises OIE as a span selection task. A span (i, j) is a continuous section of a sentence that begins with the word i and ends with the word j. The model is trained to assign the correct OIE label to each selected span, indicating whether the span belongs to a subject, predicate, or object. However, the developed sequence tagging OIE models have gaps that require further research to improve their effectiveness. More specifically, the developed models fail to address the issue of uneven BIO tag frequencies in datasets. The *O*-tag, which denotes tokens that are not part of a subject, predicate, or object, appears far more frequently than other BIO tags. For example, Stanovsky et al. [2018] observe that 58 percent of the tokens in the OIE2016 dataset are *O*-tags. As a result, the ability of the models to correctly anticipate *O*-tags has the most direct influence on the NLL. This encourages models

to place too much emphasis on O-tags rather than alternative labels found at lower frequencies. Furthermore, because the model predicts more O-tags, this could lead to shorter triple arguments.

This chapter focuses on binary relation extraction and provides many unique neural techniques to OIE. I build OIE models from three blocks: an embedding block, an encoding block, and a prediction block. I show various ways to implement each of these. Furthermore, I rigorously examine all potential combinations of their use, demonstrating that output encoding and loss function formulation have a significant impact on the results. For that purpose, I introduce and test three novel training strategies that reduce the impact of uneven tag frequencies in datasets on the models' ability to predict appropriate tags. I also investigate the impact of these novel training schemes on individual model performance.

All experiments have been conducted on the OIE16 benchmark dataset [Stanovsky and Dagan, 2016]. As part of the systematic architecture search, several existing neural architectures for OIE have been tested and compared under equal conditions.

The main contributions of this chapter are briefly summarised as follows:

- I introduce and compare different possible training schemes for OIE as a sequence tagging problem.

- I provide a large-scale study of existing and new OIE models, and compare them under the various introduced training schemes.

- I obtain the best results with a novel model based on transformers [Vaswani et al., 2017] and long short-term memories (LSTMs), and improve the current state of the art on the OIE16 benchmark dataset [Stanovsky et al., 2018] by 0.421 F_1 score and 0.420 AUC-PR, respectively.

3.2 Task Formulation

As defined earlier, I aim to extract binary relations from single sentences. Each tuple is represented as a triple $\langle$*subject*, *predicate*, *object*$\rangle$ and elements have to be non-overlapping contiguous phrases in the sentence.

Following Stanovsky et al. [2018], I treat OIE as a sequence-tagging task, where each word is labelled with a BIO tag, which indicates whether it belongs to the subject, predicate, or object. To that end, each element of a triple is implicitly extracted by labeling the according sequence of words with a **B**eginning-tag followed by an arbitrary number of **I**nside-tags, and words that are not part of any extracted phrase are marked with the **O**utside-tag. Tags are prefixed with A0 if they refer to a triple's subject, P for the predicate, and A1 to refer to the object of the extracted triple. I use a total of seven different labels, denoted as:

$$\mathcal{L} = \{A0\text{-}B, A0\text{-}I, P\text{-}B, P\text{-}I, A1\text{-}B, A1\text{-}I, O\}\,.$$

More recently, Zhan and Zhao [2020] introduced an alternative formulation, SpanOIE, that is based on spans rather than sequence tags. A span is a sub-sequence of the sentence and an extracted triple is a set of three disjunct spans, corresponding to subject, predicate, and object. They generate all possible candidate triples and use a model to score them. To restrict the exponentially growing number of possible triples, additional restrictions are put into place, such as maximum length or syntactic requirements (cf. Zhan and Zhao [2020]).

Unlike its formal counterpart, semantic parsing, the task of OIE is defined more vaguely, and hence leaves some space for interpretation. For instance, given the input sentence »*Ludwig van Beethoven was a world-famous composer of classical music.*«, one could generally consider both $\langle$*Ludwig van Beethoven*, *was*, *a world-famous composer*$\rangle$ and $\langle$*Ludwig van Beethoven*, *was*, *a world-famous composer* ***of classical***

music$\rangle$ as valid extractions from the sentence. This is illustrated in the example provided in Figure 1.1. Furthermore, there is no fixed schema of relations to be extracted. This introduces another level of complexity, as memorization of encountered patterns for a given set of relations is insufficient for a good performance and generalization.

I highlight that single sentences frequently induce more than one triple. As an example of this, consider the lower part of Figure 1.1, which illustrates a sentence that consolidates two different pieces of information.

Based on the preceding deliberations, I view any problem instance as a tuple $\langle X, Y \rangle$, where $X = \langle w_1, w_2, \ldots, w_m \rangle$ describes a sentence as a sequence of words, and $Y = \{y_1, y_2, \ldots, y_n\}$ defines a set of n valid extractions, where y_i is a sequence of BIO tags representing a single triple. Depending on the problem definition being used, the elements in Y are either sequences of BIO tags, i.e., $y_i \in \mathcal{L}^m$ for $1 \leq i \leq n$, or triples specified in terms of spans. Independently of the employed input and output formulation, the aim in this work is to model the conditional distribution $\mathbb{P}\{y \mid X\}$.

3.3 Methodology

I consider several variations and extensions of existing architectures for neural OIE viewed both as sequence-tagging and span-prediction task. To that end, I subdivide models conceptually into three blocks, which I call embedding, encoding, and prediction (listed from the bottom to the top), and investigate the impact of different NN modules for each of them. The blocks serve the obvious purposes indicated by the according designations, and are described in detail in the rest of this section.

Embedding. The embedding block represents the bottom part of the models that are considered in this chapter, and serves the purpose of mapping text to embedding vectors. The embedding block accepts text, given as sequence of tokens, and outputs a sequence of embedding vectors. Traditionally, embedding vectors were (pre-)trained

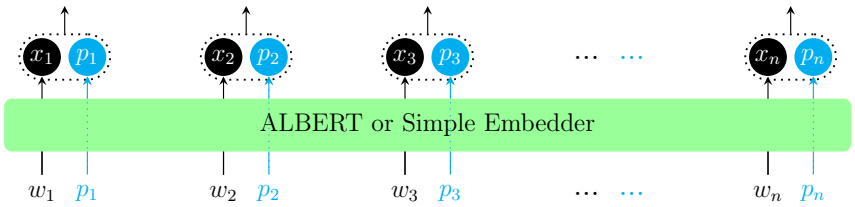


Figure 3.1: In the embedding block, each token (w_i) in the input sequence is mapped to a vector representation using the selected embedder, and the according PoS tags (p_i) are represented by means of learned embeddings. Token and PoS embeddings are concatenated and used as input to the encoding block.

for a fixed vocabulary of tokens, and used directly in place of any actual tokens in a processed input sequence [Pennington et al., 2014]. More recently, however, different models for computing contextualized vector representations of tokens in an input sequence have been used to represent text input [Alec et al., 2018, Devlin et al., 2019, Howard and Ruder, 2018], which induced notable progress on a multitude of NLP tasks.

I consider both approaches, and use either just word-piece embeddings [Wu et al., 2016], which I refer to as simple embedding block, or ALBERT [Lan et al., 2020] for computing contextualized representations. ALBERT is a variation of the well-known BERT model [Devlin et al., 2019], and makes use of the transformer [Vaswani et al., 2017] as basic building block. ALBERT, in contrast to BERT, uses parameter-reduction techniques to reduce the total number of tunable parameters in a model, resulting in lower memory consumption and increased training speed, and has been shown to perform better on several language modelling tasks. The decision to use ALBERT rather than BERT was based on the aforementioned factors.

For the task of OIE, it is a common practice to make use of PoS tags in addition to the actual input text [Stanovsky et al., 2018, Jia and Xiang, 2019, Zhan and Zhao, 2020]. To incorporate these as part of the input, I create an additional embedding vector for each PoS tag considered, and concatenate the vector representation created for a token in the input sequence with the embedding of the according PoS tag. Notice

that the embeddings of PoS tags are typically much smaller than word embeddings, and are trained jointly with the model. I follow this, and append an embedding representing the according PoS tag to every vector produced by the used embedding block. In the case of word pieces, each sub-word token is attributed the same PoS tag as the full word it belongs to. Figure 3.1 illustrates our embedding process.

Encoding. The encoding block constitutes the middle part of the considered models, which processes an embedded input sequence, as provided by the module used in the embedding block, and outputs an encoded sequence of equal length. I make use of three different NN modules for encoding embedded sequences: a BiLSTM, the encoder part of a transformer [Vaswani et al., 2017], and a BiLSTM combined with a convolutional neural network (CNN), as introduced by Jia and Xiang [2019]. For the simple BiLSTM encoder, I concatenate the top-layer hidden states of both directions, and use these as encodings of the respective tokens in the input sequence. The LSTM-CNN encoder processes a provided sequence in parallel with a BiLSTM and a CNN that are independent of each other. The used CNN consists of just one convolutional layer followed by max-pooling, and maps the entire embedded input sequence, viewed as matrix, to a single output vector. This vector is then concatenated with every step in the encoded sequence that is provided by the BiLSTM to yield the final output. Again, the encoding provided by the BiLSTM consists of the concatenated top-layer hidden states.

Note that the BiLSTM-CNN encoding block used in this work is a slightly modified version of the hybrid BiLSTM-CNN network introduced by Jia and Xiang [2019]. In the original work, the word embeddings were passed through a BiLSTM network to give $[L_f, L_b]$, where L_f is a collection of hidden states from the forward part of the BiLSTM, and L_b are the hidden states from the according backward part. In addition to this, the word representations were also passed through a CNN to compute a representation vector C. Then, all vectors in L_f and L_b as well as C

were concatenated into a single vector, and fed into a dense layer with a softmax activation to compute predictions. In this work, however, I concatenate the CNN output with every pair of forward and backward hidden-states from the BiLSTM separately, resulting in a sequence of hidden representations that is of equal length as processed input sequence. These hidden representations are then fed into the subsequent prediction block.

Prediction. The top block of a model takes an encoded sequence as input and computes a probability distribution over all possible BIO tags for each token in the encoded sequence. I consider four different model architectures as prediction blocks, three for the sequence-tagging setting, based on LSTMs, CRFs, and MLPs (illustrated in Figure 3.2), and the recently introduced SpanOIE model, which considers OIE as a span-prediction task.

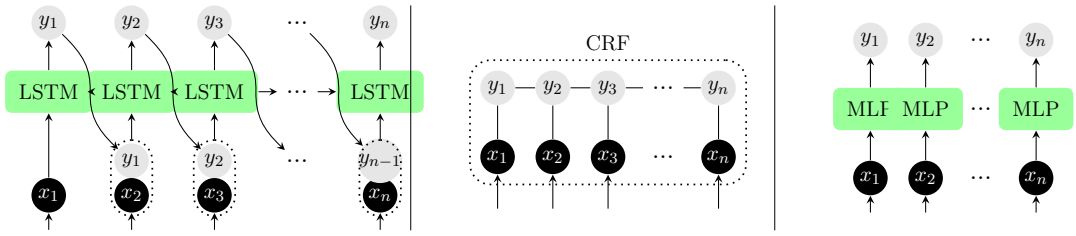
I use LSTMs for predicting label sequences from left-to-right such that every step models the conditional distribution of the next label given the encoded input sequence up to the current step as well as all previously predicted labels, i.e.,

$$\mathbb{P}\left\{\langle y_1, \ldots, y_m\rangle \mid \langle \mathbf{e}_1, \ldots, \mathbf{e}_m\rangle\right\} = \prod_i \mathbb{P}\left\{y_i \mid \mathbf{e}_1, \ldots, \mathbf{e}_i, y_1, \ldots, y_{i-1}\right\}, \tag{3.1}$$

where $\langle \mathbf{e}_1, \ldots, \mathbf{e}_m\rangle$ is an encoded sequence, as provided by the used encoding block, and $\langle y_1, \ldots, y_m\rangle \in \mathcal{L}^m$ a sequence of labels.

CRFs are commonly used in combination with sequential models, such as recurrent neural networks (RNNs), since they provide a convenient way of modeling the joint distribution of entire label sequences rather than just step-wise conditional distributions. Furthermore, using CRFs on top of a recurrent model frequently results in an increased prediction accuracy [e.g., Huang et al., 2015]. For purposes of this research, I employ a standard linear-chain CRF as prediction block.

MLPs are another family of NN modules that is frequently used for predicting



(a) (b) (c)

Figure 3.2: Illustration of the prediction block, which takes the encoded text sequence (x_i) as input, and computes a distribution over the according BIO tags (y_i). To that end, I use three different architectures based on (a) LSTMs, (b) CRFs, and (c) MLPs.

labels in the context of OIE. In contrast to the previously mentioned predictors, however, they compute labels independently for each step of an input sequence, disregarding those computed for any surrounding positions. Hence, employing MLPs is based on the simplifying assumption that

$$\mathbb{P}\left\{\langle y_1, \ldots, y_m\rangle \mid \langle \mathbf{e}_1, \ldots, \mathbf{e}_m\rangle\right\} = \prod_i \mathbb{P}\left\{y_i \mid \mathbf{e}_i\right\}, \tag{3.2}$$

using the same terminology as above.

Finally, I use the SpanOIE model as the only prediction block that is based on the span-formulation of OIE. For an encoded input sequence, this model computes scores independently for each triple in a set of candidates to extract, which are then normalized to yield a probability distribution. I refer the interested reader to Zhan and Zhao [2020] for details.

Training loss. The prediction block aims to model the probability distribution over all candidate triples to extract, as such, the NLL lends itself as the natural loss function to use. Because I regard OIE as a sequence-tagging task, optimising the NLL necessitates addressing one problematic component. Among all BIO tags, the O-tag that is used to indicate tokens that are not part of subject, predicate, or object, tends to appear at a much higher frequency than other tags. This, in turn, means that the ability to correctly predict O-tags has the strongest direct influence on the

NLL, and encourages models to focus too much on this label.

To account for this issue, I explore three novel training schemes, which disregard the probabilities of certain positions in a sequence for computing the NLL, as illustrated in Figure 3.3. The novel training schemes force the models to focus on the important parts of the OIE task, such as identifying the beginnings and ends of the subject, predicate, and object, while decreasing the importance of O-tags on the training loss A straightforward attempt to decrease the influence of O-tags on the loss term is to disregard them entirely (cf. Figure 3.3 b). Since I model probability distributions over tag sequences as opposed to computing unnormalised scores, the law of total probability allows for a network to still learn when to predict O-tags, even though it is not explicitly trained to do so. Informally, this means that $\mathbb{P}\{O\} = 1 - \mathbb{P}\{B\} - \mathbb{P}\{I\}$, which is why the trained network learns to predict O-tags when the probabilities of all remaining tags are small.

The second training scheme is based on the observation that the critical aspect of predicting a sequence of tags is identifying transitions between a block of O-tags and an element of the triple, i.e., the subject, the predicate, and the object, or directly between two of the latter. Hence, this training scheme disregards all probabilities except for those of positions that appear right before or right after a transition (cf. Figure 3.3 c). Intuitively, this makes sense, as all the disregarded tags can be determined immediately, once we know the boundaries of the subject, the predicate, and the object.

Combining the two previously outlined ideas results in our third trained scheme, which optimizes only the first and the last position of every element of the extracted triple (cf. Figure 3.3 d). Again, remaining tags can be determined from the knowledge of these tags only.

The motivation behind the novel training schemes is to alleviate the problem of imbalanced tag occurrences in sequences (O-tags typically appear most frequently),

(a)	*O*	*O*	*A0-B*	*A0-I*	*A0-I*	*O*	*O*	*O*	*O*	*P-B*	*P-I*	*O*	*A1-B*	*A1-I*	*O*
(b)	*O*	*O*	*A0-B*	*A0-I*	*A0-I*	*O*	*O*	*O*	*O*	*P-B*	*P-I*	*O*	*A1-B*	*A1-I*	*O*
(c)	*O*	*O*	*A0-B*	*A0-I*	*A0-I*	*O*	*O*	*O*	*O*	*P-B*	*P-I*	*O*	*A1-B*	*A1-I*	*O*
(d)	*O*	*O*	*A0-B*	*A0-I*	*A0-I*	*O*	*O*	*O*	*O*	*P-B*	*P-I*	*O*	*A1-B*	*A1-I*	*O*

Figure 3.3: (a) Standard way of computing the loss where all tags are considered. (b) Excluding all *O* tags when computing the loss. (c) Considering only the tags in the transitions between different tags (d) Considering only the tags in the transitions between different tags that are not *O*. (b), (c), and (d) illustrate our three novel training schemes.

which leads to models that predict too many *O*'s. I avoid this problem by ignoring (some of) them when computing a loss term. However, keep in mind that a model can still learn about *O*'s because it is trained to compute a probability distribution over all tags for each position in a sequence. As a result, if the loss provides a training signal for predicting Probability(B) and Probability(I), it also provides an implicit signal for predicting Probability(O) = 1 - Probability(B) - Probability(I).

Finally, notice that the presented schemes are used during the training of a model only. For inference and sampling, I make use of all probabilities, including those of *O*-tags.

3.4 Experiments, Evaluation, Results, and Analysis

Dataset. For experiments, I made use of the OIE16 benchmark dataset [Stanovsky and Dagan, 2016]. With a total of 5,078 training samples, the dataset is rather small, though, making it hard to train models that generalize to unseen problem instances. For this work, I thus augmented the OIE16 training data with samples from another dataset created by Cui et al. [2018]. The latter is a huge dataset, consisting of more than 36M training samples that have been generated using OpenIE4 [Angeli et al., 2015], and contains examples with lower quality than the OIE16 data. Furthermore, there is usually just one target triple to extract per sentence in this dataset, while OIE models are generally expected to find all of them.

To make effective use of the low-quality dataset, I had to perform a number of preprocessing steps. First and foremost, some samples make use of additional tags for specifying label sequences. Following Cui et al. [2018], I discarded these tags, which results in samples that are still valid and compatible with the standard set of BIO tags, as presented above. Furthermore, Angiras [2018] observed that models trained on the huge low-quality dataset tend to saturate in the first training epoch already, usually resulting in low training, but high test error. To obtain better results, I thus filtered the dataset with respect to the distribution of predicates in the target triples. I observed that just about 8K out of more than 71K predicates appear at least 50 times, and pruned all training samples with predicates below this threshold. This resulted in a dataset consisting of about 28M sentences. Next, I observed highly uneven occurrence statistics of the remaining predicates, and, following Cui et al. [2018], further down-sampled the data to avoid training on a dataset with highly imbalanced target predicates. This down-sampling, on the other hand, could not be done at random. According to the distribution of predicate heads across the sentences, the most frequently occurring predicate word appears in 4,852,649 training examples, the 100-th most common one appears 31,999 times, and the 500-th most common one appears only 7,484 times. Furthermore, the 100 most common predicates alone account for 16,376,985 training examples. If I randomly select only a small number of training examples from this distribution, I may end up with a training set with a very skewed representation of the predicates. To that end, I proceeded as follows:

- for predicates appearing less than 500 times, I sampled 50 examples,
- for those with 500 to 1K occurrences, I selected 200 examples,
- predicates with 1K to 10K occurrences were down-sampled to 500 examples,
- for predicates with 10K and 100K occurrences, I sampled 1K examples, and
- for all predicates that appear more than 100K times, I sampled 3K examples.

Sampling was performed uniformly at random, and reduced the training data to a total of 1.7M training samples, which were combined with the training partition of the OIE16 benchmark dataset. Although this new training dataset retains the essence of the original dataset's distribution of predicates, it is less skewed and ensures a good representation of all predicates.

Experimental evaluation. I conducted a large-scale study in which I trained and evaluated all combinations of embedding, encoding, and prediction blocks that I introduced above. In all cases, the validation accuracy was determined using the validation data set provided by Stanovsky and Dagan [2016]. For those models containing an ALBERT embedding block, I made use of a pretrained ALBERT model (`albert-base-v1`) provided by Wolf et al. [2020]. Due to resource constraints, I had to freeze the parameters of the ALBERT blocks to speed up the training of our models. For models containing a simple embedding block, I used word-piece embeddings [Wu et al., 2016] rather than word embeddings, since these yielded better results in the experiments, and employed the embedding vectors that were pretrained with the same ALBERT model used in the ALBERT embedding blocks to initialize the model.

Notice that the architectures considered in this study also cover all the currently most important methods of OIE that are based on deep learning (DL) (Angiras, 2018; Cui et al., 2018; Stanovsky et al., 2018; Zhan and Zhao, 2020; and a slightly modified version the model by Jia and Xiang 2019, as described in Section 3.3). I do not compare to any rule-based OIE systems, though, as they have been shown to consistently achieve inferior results than DL-based approaches in related work [Angiras, 2018, Cui et al., 2018, Stanovsky and Dagan, 2016, Stanovsky et al., 2018].

Models that use either the LSTM or the MLP prediction block were trained and evaluated for all the training schemes presented above. Other predictors cannot be used with any of the newly introduced schemes, though, and were thus trained by minimizing the standard NLL only.

All considered models were evaluated on the validation partition of the OIE16 dataset with respect to both F_1 score and the AUC-PR. To that end, I computed the top-20 predictions for each of the samples in the test data, using either beam search or, in the case of the CRF predictor, the Viterbi algorithm, and considered all predictions with a probability above a certain threshold as extractions of the model evaluated. As the prediction threshold, I chose the one that achieved the maximum F_1 score on the validation partition of the OIE16 dataset separately for each model. Notice that the prediction threshold was well below 0.5 in all our experiments, usually around 0.1, and thus allowed for extracting multiple triples per sample.

To determine whether a predicted triple matches any of the target triples in the test data, I employed the evaluation scheme that is typically used in the context of OIE [He et al., 2015, Stanovsky and Dagan, 2016].[1] To that end, a predicted triple is considered correct, if each of its elements, i.e., subject, predicate, and object, contains the syntactic head of the corresponding element in the target triple. For instance, if the test set contains the triple $\langle$*Donald Trump*, *is*, *president of the U.S.*$\rangle$, then the prediction $\langle$*Trump*, *is*, *president of the U.S.*$\rangle$ is considered correct, as »*Trump*« is the syntactic head of the subject phrase »*Donald Trump*«.

Occasionally, OIE models are evaluated with predicate-head hinting [Stanovsky et al., 2018], which means that the beginning of the predicate in the target triple is marked as part of the input. More precisely, a special token is inserted into the input sequence right before the first token that is part of the predicate to extract, the so-called predicate head. This, obviously, makes the task of OIE considerably easier, and does not reflect the typical problem scenario. Nevertheless, I run all experiments a second time with predicate-head hinting, and provide the according results in Appendix A.1.

Due to the great number of experiments that have been conducted, it was not

[1] This is the same evaluation scheme that is used by the well-known benchmark script by Stanovsky and Dagan [2016].

possible to perform grid search for every one of them separately. Instead, I chose a set of hyperparameters that I found to work well across a multitude of initial training runs, and kept them constant over all performed experiments. The exact hyperparameter values are reported in Appendix A.2.

Results. Table 3.1 summarizes the results of the comparative study that I carried out, and provides a number of interesting insights. First and foremost, it can be observed that the model with the transformer encoding and the LSTM prediction block achieved the best F_1 score both with and without ALBERT embedding block. The prior is also the overall best model with respect to both F_1 score and AUC-PR, and outperformed all the considered state-of-the-art approaches by a significant margin of at least 0.421 F_1 score and 0.420 AUC-PR, respectively. While one might have expected to see the transformer encoding block at the top of the score board, it is surprising that the CRF predictor performed significantly worse than the LSTM and MLP prediction blocks, as CRFs have previously achieved strong results on different task of sequence prediction [e.g., Huang et al., 2015]. What comes less at surprise, though, is that using an ALBERT embedding block led to an increased performance in almost all cases, which is in line with the existing research on BERT and related models. Another important insight is that the newly introduced training schemes for sequence-tagging models helped to boost performance significantly in comparison with the usual way of computing the training loss. In this context, optimizing the first and the final tokens of subject, predicate, and object only proved particularly useful (column (d) in the table), and led to the best results for almost all encoding and prediction blocks.

Note that the results that I have achieved for models from related work differ significantly from those reported in the respective papers, which is easily explained by the difference in how models were evaluated in the same. In the context of OIE, it is common practice to pre-select candidate triples during inference, and subsequently

use a model to score each of them. While this might make sense for optimizing a system in a production environment, it obfuscates a model's true ability to some extent, which is why I decided to not use any kind of pre-selection at all.

			LSTM				MLP				CRF	Span
			(a)	(b)	(c)	(d)	(a)	(b)	(c)	(d)	(a)	(a)
(1)	**None**	F_1	0.325	0.402	0.320	0.411	0.125	0.224	0.225	0.325	0.199	0.223
		AUC-PR	0.184	0.172	0.181	0.178	0.113	0.112	0.1110	0.114	**0.136**	0.177
	BiLSTM	F_1	0.211	0.535	0.346	0.598	0.123	0.276	0.213	0.460	0.181	0.221
		AUC-PR	0.140	0.524	0.277	0.602	0.055	0.219	0.213	0.435	0.119	**0.195**
	BiLSTM-CNN	F_1	0.225	0.486	0.588	0.589	0.113	0.250	0.241	0.452	0.201	0.230
		AUC-PR	0.112	0.474	0.574	**0.610**	0.065	0.198	0.196	0.420	0.124	**0.189**
	Transformer	F_1	0.332	0.539	0.351	**0.601**	0.126	0.281	0.260	**0.471**	**0.205**	**0.242**
		AUC-PR	0.132	0.534	0.321	0.597	0.070	0.183	0.206	**0.439**	0.117	0.178
(2)	**None**	F_1	0.329	0.406	0.323	0.415	0.126	0.226	0.225	0.326	0.203	0.256
		AUC-PR	0.195	0.175	0.193	0.187	0.107	0.106	0.105	0.107	0.145	0.187
	BiLSTM	F_1	0.333	0.541	0.349	0.623	0.161	0.281	0.263	0.463	0.185	0.253
		AUC-PR	0.250	0.504	0.286	0.610	0.120	0.220	0.201	0.435	0.107	**0.205**
	BiLSTM-CNN	F_1	0.186	0.463	0.596	0.610	0.123	0.276	0.258	0.468	0.203	0.253
		AUC-PR	0.0.018	0.397	0.582	0.614	0.054	0.219	0.211	0.431	0.131	0.193
	Transformer	F_1	0.351	0.555	0.362	**0.628**	0.117	0.292	0.274	**0.476**	**0.217**	**0.273**
		AUC-PR	0.242	0.515	0.278	0.644	0.044	0.204	0.218	**0.436**	**0.149**	0.198

References: Angiras [2018] Stanovsky et al. [2018] Jia and Xiang [2019] Zhan and Zhao [2020]

Table 3.1: The results of the experimental evaluation, where the different columns correspond with the different training schemes: (a) standard NLL (i.e., considering all labels), (b) disregarding O-tags, (c) optimizing transitions only, and (d) considering start and end of a triple's elements only. The best results are underlined for each of the encoding blocks, and printed boldface for each prediction block. Furthermore, **(1)** represents models trained with a simple embedding block, and **(2)** represents models trained with ALBERT embedding block.

Analysis and ablation studies. In this section, I further analyze the results presented above, including the ablations that have been performed as part of our comparative study already. Furthermore, I present additional ablation studies for the best-performing model.

At the bare minimum, a model consists of a simple embedding block as well as one of the prediction blocks presented above, while encoding blocks are generally

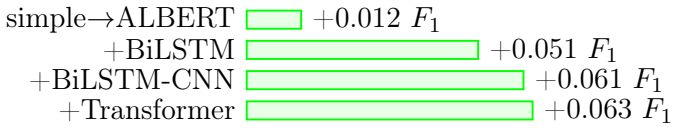


Figure 3.4: The average improvements caused by using an ALBERT embedding block in place of a simple one and by adding an encoding block to a model in terms of F_1 score, computed across all experiments performed.

optional. Hence, I first investigate the effect of moving from a simple to an ALBERT embedding block on the one hand and how adding an encoding block influences a model's performance on the other—Figure 3.4 summarizes insights from this analysis. First and foremost, I observed that encoding blocks, on average, cause much higher improvements than using an ALBERT embedding block in place of a simple one. More precisely, ALBERT caused just a small mean improvement of 0.012 F_1. Encoding blocks, however, pushed the performance notably by 0.058 F_1, on average, and, as expected, transformers performed best among all considered encoding blocks.

Next, I compare the various prediction blocks that were used in the experiments. To that end, Figure 3.5 illustrates the mean performance of each prediction block contrasted with the mean F_1 score computed over all experiments performed. Surprisingly, the LSTM prediction block preformed, on average, by far best among all predictors considered, and outperformed all other prediction blocks for each of the training schemes considered. Another surprise is that even the MLP predictor achieved better results than the CRF prediction block for all training schemes except (a). This contrasts previous findings on the use of CRFs for sequence-tagging [Huang et al., 2015], and suggests that the training scheme, which is analyzed next, is a much bigger impact on a model's performance than certain choices about its architecture for the task of OIE. Therefore, when trained with proper loss formulation, a simple predictor, such as an MLP, can outperform a more powerful one, such as a CRF, which does not allow for using the introduced training schemes. Finally, the mean performance of the SpanOIE prediction block was found to be in between those values observed

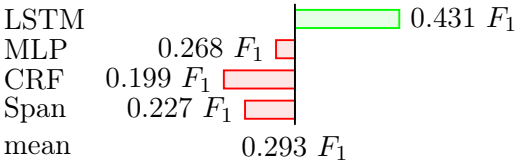


Figure 3.5: The average F_1 score achieved by each of the different prediction blocks in comparison with the overall mean F_1.

for the CRF and the MLP predictor.

Finally, Figure 3.6 compares the mean performance achieved for each of the different training schemes with the overall mean F_1 score computed over all experiments with sequence-tagging models (as schemes (b) to (d) can be used with these models only). As illustrated in the figure, I observed significant differences among the training schemes. First and foremost, it can be observed that training scheme (d), which optimizes the first and last positions of subjects, predicates, and objects only, clearly outperformed all other schemes. Intuitively, this makes sense, as this view reduces the problem of OIE to the absolute minimum, which is the question of where elements start and end, respectively. In contrast to this, optimizing the NLL on entire label sequences, i.e., scheme (a), led to the worst mean performance. This suggests that O-tags, which appear most frequently among all labels, are attributed too much importance in general, and steer away attention from important aspects such as the boundaries of triples' elements, which are emphasized by the other training schemes. Schemes (b) and (c) resulted in similar mean F_1 scores, close to the overall mean performance, and hence performed significantly worse than (d). Since the parts of a label sequence considered by each of these three schemes allow for reconstructing the entire sequence, one possible explanation for this is, once again, that training scheme (d) reduces the task to the absolute minimum, which might also render the according learning task as easy as possible.

I want to emphasize that the lenient evaluation scheme that is used in the context of OIE allows models to »*cheat*« in the sense that they can learn to not predict O-tags

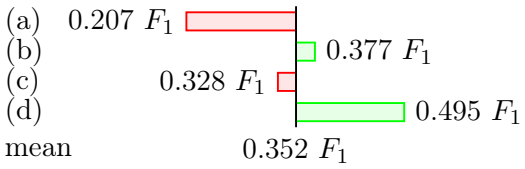


Figure 3.6: The average F_1 score achieved by each of the different training schemes, computed just for sequence-tagging models in comparison with the overall mean F_1 for the according experiments.

at all to improve the chances that very long subjects, predicates, and objects actually cover the syntactic heads of some target triples. To show that this does not happen with the models in this study, I computed how often the top model predicted an *O*-tag correctly (as *O*-tag) and how many it labelled as part of subjects, predicates, and objects, respectively. Table 3.2 summarizes the findings, and shows that the model is highly accurate on subjects and predicates, with less than one *O*-tag prepended and appended to any target subject or predicate on average. For objects, I find similar numbers for prepended, but slightly higher ones for appended *O*'s. Manual inspection of the data reveals, however, that this is justified in most cases. A good example for this is the one that has been used above: if the target object »*a composer*« is predicted as »*a composer of classical music*«, then »*of classical music*« is a suffix that is represented as series of *O*-tags in the dataset, but which may very well be considered as part of the object. Overall, the model predicted on average 78.7% of all *O*-tags correctly as such for each sample in the test data, which together with the other statistics supports the conclusion that it did not learn to cheat by generating overly long elements.

For the best-performing model, I conducted additional ablation studies, looking at how its performance changes when the number of layers in the encoder (1 in the top model) and the used hidden-layer size, in both encoder and predictor, (512 in the top model) is varied, as summarized in Table 3.3. To that end, I noted that increasing the number of layers in the transformer encoding block resulted in lower values of both

		Training Scheme			
		(a)	**(b)**	**(c)**	**(d)**
Subject	**Prefix**	0.60	0.49	0.50	0.50
	Suffix	0.51	0.43	0.44	0.52
Predicate	**Prefix**	0.01	0.01	0.01	0.01
	Suffix	0.03	0.03	0.03	0.04
Object	**Prefix**	0.63	0.61	0.59	0.62
	Suffix	4.23	4.11	3.71	4.55

Table 3.2: The average number of O-tags predicted as part of subject, predicate, and object, respectively. To that end, I refer to O-tags that were prepended to a target element as prefix, and those that were appended as suffix.

Enc. Layers	**Hidden Size**	**Score**	**Training Scheme**			
			(a)	**(b)**	**(c)**	**(d)**
6	128	F_1	0.360	0.402	0.306	0.426
		AUC-PR	0.200	0.201	0.196	0.213
1	128	F_1	0.347	0.428	0.301	0.431
		AUC-PR	0.212	0.303	0.223	0.209
1	**512**	F_1	0.351	0.555	0.362	**0.628**
		AUC-PR	0.242	0.515	0.278	**0.644**
1	1024	F_1	0.418	0.424	0.426	0.446
		AUC-PR	0.195	0.204	0.206	0.223

Table 3.3: Results achieved for different configurations of our best-performing model, which consists of an ALBERT embedding block, a transformer encoding block, and an LSTM prediction block. The best configuration is printed boldface.

F_1 score and AUC-PR. The same was the case for both increasing and decreasing the used hidden size. Since the training data consists of a total of about 1.7M samples only, a likely explanation is that there is balance between under- and overfitting the data, which the top model seems to address well. Finally, I notice a similar pattern with respect to the different training schemes as the one discussed above.

Some of the existing works on neural OIE employ a correction of malformed predicted label sequences [Stanovsky et al., 2018]. To that end, orphan intermediate labels, i.e., ones that are preceded neither by the according head nor by the same intermediate label, are corrected to O-tags. In experiments under this work, this approach did not lead to any improvements, and is thus not further elaborated on.

Finally, Appendix A.1 provides additional insights gained in the experiments with predicate-head hinting, which led to an average improvement of 0.074 F_1 score and 0.077 AUC-PR, respectively.

In general, I believe there is a trade-off between selecting the right model architecture and developing an appropriate problem/loss formulation. However, one of the lessons that this chapter emphasises is that loss functions must be designed in such a way that they truly emphasise the key aspects of the considered task. The ability to identify the beginnings and ends of the subject, the predicate, and the object, for example, is a key aspect in OIE. The results presented in this chapter demonstrate that models trained with the novel loss formulations can identify key aspects of OIE and reduce the importance of O-tags on the training loss.

Significance of the training schemes. I had to assess the statistical significance of the improvements arising from training schemes (b), (c), and (d) over the traditional NLL (a) to ensure that the innovative training schemes do not simply outperform the traditional NLL by chance. To accomplish this, I used a paired t-test to see if there were any variations in the F_1 and AUC-PR values obtained by the models between each of the new schemes and the default NLL based on the results presented in Table 3.1. I am only interested in models where all these training schemes can be used with these models only, namely, models with either an LSTM or MLP in the prediction block.

I began by comparing the training scheme (b) with the default NLL. When computing the loss, the training scheme (b) excludes all O-tags. In both cases, the paired t-test on the F_1 and the AUC-PR values yielded p values of less than .05 ($p < .05$). This provides solid evidence that the default NLL has a less than 5% chance of outperforming the training scheme (b). Therefore, the training scheme (b) outperforms the conventional NLL significantly. Second, the paired t-test of F_1 and AUC-PR values from training schemes (c) and (a) yielded p values of less

than .03 ($p < .03$). This provides solid evidence that the default NLL has a less than 3% chance of outperforming the training scheme (c). To that end, the training scheme (c) outperforms the conventional NLL significantly. Second, the paired t-test of F_1 and AUC-PR values from training schemes (c) and (a) resulted in p values of less than .03 ($p < .03$). This shows that the default NLL has a slim possibility of outperforming the training scheme (c) by less than 3%. For that purpose, the training method (c) greatly outperforms the traditional NLL. Finally, p values smaller than .01, $p < .01$, were obtained using the paired t-test of F_1 and AUC-PR values from training schemes (d) and (a). This reveals that the default NLL only has a 1% chance of outperforming the training scheme (d). The training approach (d) outperforms the classic NLL significantly for this purpose.

Based on the results of the aforementioned investigation, I can conclude that each unique training scheme outperforms the classic NLL. As a result, I conclude that the gains resulting from the novel training strategies are significant and not purely coincidental.

3.5 Related Work

Recently, several approaches based on NN architectures have been introduced [Stanovsky et al., 2018, Cui et al., 2018, Zhan and Zhao, 2020, Jia and Xiang, 2019]. As one of the first neural methods, Stanovsky et al. [2018] treat OIE as a sequence-tagging task. Their approach uses stacked layers of BiLSTMs with a Softmax layer. The layers of BiLSTMs compute feature vectors and the Softmax layer predicts a BIO tag for each word in the input sentence. Jia and Xiang [2019] use the same sequence-tagging problem formulation as Stanovsky et al. [2018] and extend their work with systematic tests of various NN architectures, such as (Bi)LSTMs, CNNs, and their combinations with CRFs. Finally, they develop a hybrid model that

combines BiLSTMs, CNNs, and CRFs to achieve a maximal performance. Zhan and Zhao [2020] introduce a span model for n-ary OIE to replace the previously adopted sequence-tagging formulation. Their BiLSTM-based model finds predicate spans separately and uses a separate BiLSTM module to find arguments (entities), given a predicate as an input.

Unlike prior research, this work addresses the problem of uneven tag frequencies in datasets. This is accomplished by developing and testing three innovative training strategies that mitigate the influence of unequal tag frequencies in datasets. Furthermore, as a result of various NN combinations in the embedding, encoding, and prediction blocks, I develop a number of novel OIE strategies.

3.6 Discussion

In this chapter, I have presented novel training approaches for mitigating the influence of uneven tag frequencies on the models' ability to predict appropriate tags. I have carefully investigated a variety of different NN architectures for OIE, as well as training schemes. In doing so, I have improved the SOTA on the OIE16 benchmark by 0.421 F_1 and 0.420 AUC-PR, respectively, with a novel model composed of an ALBERT embedding block, a transformer encoding block, and an LSTM prediction block, all of which were trained using a training scheme that used a newly introduced loss formulation. Following a thorough investigation, it was discovered that picking the proper training scheme is just as crucial as selecting the neural model architecture, because the conventional NLL loss gives too much weight to non-essential elements of the data and repeatedly produces poor results.

Chapter 4

Real or Fake? Discriminative Training for Open Information Extraction with Syntactically Rich Embeddings

In this chapter, I look at OIE as a sequence generation problem. OIE is interested in extracting facts of the form "(Subject, Relation, Object)" from natural language text. Several new strategies for training neural OIE models are proposed in this work. To begin, I offer a novel method for computing syntactically rich embeddings that is guided by the structure of dependency trees. Second, I provide mechanisms to restrict repeating tokens in extracted facts, as well as improve models' ability to generate implicit facts. Finally, I suggest a novel discriminative strategy for training neural OIE models.

4.1 Introduction

Several neural OIE approaches have been proposed that treat OIE as either a sequence labelling problem [Stanovsky et al., 2018, Roy et al., 2019, Jiang et al., 2019, Zhan and Zhao, 2020] or a sequence generation problem [Cui et al., 2018, Sun et al., 2018, Kolluru et al., 2020]. Sequence labelling approaches classify each token in the input text into one of three categories: subject, relation, or object, whereas sequence generation approaches generate facts one word at a time from the input text.

For the task of OIE, it is a common practice to make use of PoS and dependency tags in addition to the actual input text as a way of incorporating syntactic information [Stanovsky et al., 2018, Zhan and Zhao, 2020, Jia and Xiang, 2019, Sun et al., 2018]. In such work where these tags are used, the embeddings for the tags are just concatenated to the embeddings of the corresponding text tokens. This formulation does not fully use the rich syntactic information, especially the one that is expressed in the structure of dependency trees. Dependency trees' head-dependent relations provide a good approximation of the semantic relationships between predicates and their arguments [Jurafsky and Martin, 2009]. As a result, they are directly applicable to a wide range of applications, including IE. In this work, I compute token representations based on the structure of dependency trees in order to benefit from their rich syntactic and semantic information. Furthermore, sequence generation approaches are susceptible to generating facts that often express redundant information and are also prone to generating repetitive text in facts. Sun et al. [2018] and Kolluru et al. [2020] looked into the issue of generating the same facts multiple times, redundant information. No work has been done to control the generation of repetitive text in facts. Additionally, sequence generation approaches are capable of introducing words that are in the vocabulary but not in the input sentence into a generated fact. This capability enables such approaches to generate facts that are implicitly stated in the input sentence. However, the approaches employed so far by Cui et al. [2018]

and Sun et al. [2018] are restrictive in how the models "pick" words when generating facts, which restrains the models' flexibility in generating implicit facts.

The approach in this work uses a sequence generation approach to generate facts from natural language text a word at a time. This approach, unlike the previous approaches, computes syntactically rich vector representations of input text tokens guided by the structure of dependency trees. For each input sentence, I construct a visibility matrix of its tokens based on the structure of its dependency tree. I consider tokens that are directly related in the dependency tree as visible to each other. I then feed the token embeddings and the visibility matrix into a GNN [Zhou et al., 2020] encoder that computes new token embeddings guided by the visibility matrix. In addition to computing syntactically rich vector representations, I also introduce a new method for training neural OIE models. I add an extra module, the discriminator, that takes the generated tuple as input and classifies its tokens as either "real" or "fake", where "real" tokens are those that are in both the generated and the gold tuples, while "fake" tokens are those that are in the generated tuple but not in the gold tuple. Figure 4.1 illustrates my approach. To ensure that such a model does not generate repetitive text, I use a coverage vector to monitor the degree of coverage that words in the input text have received so far. When generating the next word at timestep t, the coverage vector at that instant is a sum of all attention distributions computed over the input text tokens from timesteps 0 to $t-1$. This coverage vector is used as part of the input when computing the next attention distribution. Intuitively, this informs the current attention mechanism's decision of the previous decisions and makes it easier to avoid repeatedly attending to the same words in the input text, hence avoiding generating repetitive text in a fact. As a means to explicitly guide the model's choice of "selecting" a word from either the vocabulary or the input text, I explicitly compute the probability of generating a word from either the vocabulary or input text using the model's context vectors. Lastly, I also investigate the effect

of data-augmentation on the performance of our models.

The main contributions of this chapter are briefly summarized as follows:

- I present several new methods for training neural OIE models. First, I propose a new method of computing syntactically rich vector representations of input tokens guided by the structure of dependency trees. This is done by using GNNs, as they are able to take into account the graph structure of a dependency tree.

- Second, I propose an additional module, the discriminator, on top of the model that generates facts. The additional module performs a binary classification of tokens in generated facts as either "real" or "fake". This new approach significantly improves the performance of sequence-to-sequence neural OIE models.

- Furthermore, I reduce repetitive words in generated facts by 23%, from 36% to 13%, when I jointly use the coverage vector and explicitly guide the model where to pick the next word when generating a fact.

- Finally, I present paraphrased versions of the CaRB [Bhardwaj et al., 2019], OIE2016 [Stanovsky and Dagan, 2016], and LSOIE [Solawetz and Larson, 2021] datasets, which I use to augment existing datasets.

- The best performing model uses both the discriminator and the GNN encoder, and beats the SOTA of IMoJIE on CaRB, with an improvement of 39.63% in F_1 score. This model also presents competitive results on OIE2016 and LSOIE.

4.2 Task Formulation

In this work, I approach OIE as a sequence generation task. As illustrated in Figure 4.1, given an input text, the generator produces a tuple one word at a time. Then, I pass the generated tuple through the discriminator, which classifies tokens in the tuple as either "real" or "fake", where "real" tokens are those that are in both

the generated tuple and the gold tuple, while "fake" tokens are those that are in the generated tuple but not in the gold tuple. Thus, the discriminator performs binary classification of the tokens in the generated tuple. I only consider binary extractions from single sentences.

Given a sentence-tuple pair $\langle X, Y \rangle$, where$X = \langle w_1, \ldots, w_n \rangle$ and $Y = \langle y_1, \ldots, y_m \rangle$ are sequences of tokens in the input sentence and expected tuple, respectively, I define the generator's conditional probability of $P(Y|X) = \mathbb{P}\{Y|\langle w_1, \ldots, w_n \rangle\}$ as

$$\prod_i \mathbb{P}\left\{y_i \mid \langle w_1, \ldots, w_n \rangle; \langle y_1, \ldots, y_{i-1} \rangle\right\}, \tag{4.1}$$

where $P(y_i|\langle w_1, \ldots, w_n \rangle; \langle y_1, \ldots, y_{i-1} \rangle)$ is the probability of generating the i-th word, given the input sequence and the entire generated sequence by step i, and $P(Y|X)$ is the probability of generating an entire tuple Y, given the input sentence X.

Given a generated tuple, $Y = \langle y_1, \ldots, y_m \rangle$, and the input context vector, d^*, the discriminator seeks to label each token in Y as either "real" or "fake". The input context vector is weighted average of the vector representations of the input sequence. To that end, I define the discriminator's likelihood $P(L|Y)$ as

$$\prod_{i=1}^{m} [\, \sigma(v^T[\mathbf{y}_i, \mathbf{d}^*])^{(y_i^{real} = y_i)} \, (1 - \sigma(v^T[\mathbf{y}_i, \mathbf{d}^*]))^{(y_i^{real} \neq y_i)}], \tag{4.2}$$

where $L = \{0, 1\}$ with 1 and 0 standing for "real" and "fake" tokens, respectively, σ is the sigmoid function, $\mathbf{y}_i$ is the vector representation of the i-th token y_i in the generated tuple, $\mathbf{d}^*$ is the input context vector, v is a set of learnable parameters, and $[,]$ represents the concatenation operation.

4.3 Methodology

In this section, I define and describe the modules that I designed to solve OIE as defined in Section 4.2. Figure 4.1 illustrates the approach that used in this research.

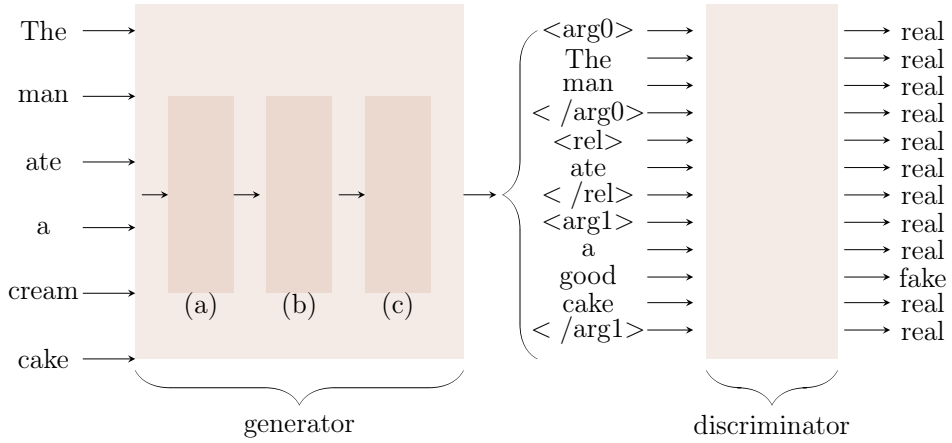


Figure 4.1: The approach consists of a generator and a discriminator. The generator produces a tuple from an input sentence. The discriminator takes the generated tuple and predicts whether each of its tokens is either "real" or "fake": "real" tokens are tokens that are present in both the generated tuple and the target tuple, while "fake" tokens are tokens that are in the generated tuple but not in the target tuple. The generator has three components: (a) the embedding block, (b) an optional encoding block, and (c) the decoding block.

4.3.1 Embedding

The embedding block maps text, pre-processed into a sequence of tokens, to a corresponding sequence of embedding vectors. In this chapter, I consider stacked layers of BiLSTMs, a transformer encoder [Vaswani et al., 2017], pre-trained BERT [Devlin et al., 2019], a feedback transformer encoder [Fan et al., 2020], and a pre-trained ELECTRA model[1] [Clark et al., 2020] for the embedding block. Unlike transformers in which the representation at a given layer can only access representations from lower layers rather than the higher level representations already available, feedback trans-

[1]https://huggingface.co/google/electra-small-discriminator

formers expose all previous representations to all future representations [Fan et al., 2020]. That is, the lowest representation at the current timestep t is formed from the highest-level representations of the past. As such, feedback transformers have an improved capacity to capture the sequential property of the input. Unlike the transformer, which does not fully use the input's sequential property, since all input tokens are processed in parallel, feedback transformers specialise in sequential token prediction, which is essential for tuple generation. Unlike other pre-trained language models that are trained through MLM, such as BERT, ELECTRA is pre-trained by detecting replaced tokens in the input sequence.

For a given sequence of input tokens, the embedding block computes a corresponding sequence of embedding vectors of the same length as the input sequence. In addition to input text, I also incorporate PoS and dependency tree tags obtained via the Spacy library[2]. I separately compute embedding vectors for each PoS and dependency tree tag, and concatenate them to the vector representation of the corresponding input text token. For words that are split into subwords, each subword token is attributed the same PoS and dependency tree tag as the parent word that it belongs to.

4.3.2 Encoding

The encoding block computes vector representations of the input sequence guided by the structure of the dependency tree. I construct a visibility matrix of tokens in the input sequence based on the proximity of tokens in the corresponding dependency tree. The encoder takes a sequence of vector representations from the embedding block and the visibility matrix as input, and gives a sequence of vector representations of the same length as the input sequence as output. The encoding block is optional. If the encoding block is not used, the representations from the embedding block are passed

[2] `https://spacy.io/usage/linguistic-features`

directly to the decoding block.

To take into account the graph structure of the dependency tree when computing vector representations, I use a GAT [Veličković et al., 2018] as encoder. GATs compute the vector representation of the current node by only attending to nodes in its neighborhood as defined in the visibility matrix. In this case, I treat each token in the input sequence as a node and those that it is connected to in the dependency tree as its neighbors. Thus, the vector representation of the current node is computed by only attending to the vector representations of the nodes in the neighborhood as presented in the visibility matrix. For comparison, I also use a transformer encoder that does not take into account the visibility matrix when computing vector representations. The transformer encoder attends to all tokens in the input sequence when computing vector representations.

4.3.3 Decoding

During decoding, I adopt pointer-generator-networks (PGNs) [See et al., 2017] and make modifications. For example, I calculate the distribution over the entire vocabulary from a weighted sum of the model's and decoder's context vectors, as opposed to the concatenation of the two. When generating the next token in a fact at timestep t, the embedding vectors of the entire generated sequence so far, $\langle y_1, \ldots, y_{t-1} \rangle$, are fed into either the BiLSTM network, a transformer, or feedback transformer to generate contextualized decoder hidden representations.

The decoder hidden representations $\langle d_1, \ldots, d_{t-1} \rangle$ are then used to calculate the decoder context vector, d_t^*, which is a weighted sum of decoder hidden representations [Bahdanau et al., 2015]:

$$e_i^t = v^T \, Tanh(W_h d_i + b_{attn}) \,,$$

where v, W_h, and b_{attn} are learnable parameters, and d_i are decoder hidden representations. Then, e^t is used to calculate attention scores:

$$a^t = Softmax(e^t). \tag{4.3}$$

The attention scores at timestep t, a^t, are used to calculate the decoder context vector d_t^* as weighted sum of decoder hidden representations:

$$d_t^* = \sum_{i=1}^{t-1} a_i^t d_i, \tag{4.4}$$

where d_i are decoder context vectors, and a_t are attention scores over decoder context vectors at timestep t. The computed decoder context vectors capture are longer context, $\langle d_1, \ldots, d_{t-1} \rangle$, unlike in previous pieces of work where they only considered the previously decoded token, d_{t-1}.

Coverage mechanism. I use a coverage vector, c_t, to keep track of how much attention each token in the input text has received while generating a tuple one word at a time. This avoids repeatedly attending to the same tokens in the input text. The coverage vector is the sum of all attention distributions over encoder hidden representations up to timestep $t-1$:

$$c^t = \sum_{t'=0}^{t-1} \alpha^{t'},$$

where $\alpha^{t'}$ is the attention distribution over encoder hidden representations. The coverage vector is initialized as a zero vector, as on the first timestep, no input token has been attended to.

Decoder-encoder attention. The decoder context vector d_t^* from Eq. 4.4, the encoder hidden representations from Section 4.3.2, and the coverage vector c^t are used to compute the model's context vector h_t^*, using attention [Bahdanau et al.,

2015]:

$$e_i^t = v^T Tanh(W_h h_i + W_d d_t^* + W_c c^t + b_{attn}), \tag{4.5}$$

where v, W_h, W_d, W_c, and b_{attn} are learnable parameters, h_i are encoder hidden representations, and c^t is the coverage vector. The coverage vector is included in this computation in Eq. 4.5 so that the attention mechanism is informed of previous attention decisions, hence avoiding repeatedly attending to the same words in the input text. The attention distribution over input tokens, α^t, is calculated from e^t using Eq. 4.3. This α^t and encoder hidden states, $\langle h_1, \ldots, h_n \rangle$, are then used to compute the model's context vector h_t^* using Eq. 4.4.

The attention distribution α^t obtained here can be interpreted as a probability distribution over the source words, which tells the model where to look to produce the next word.

To compute the distribution over the entire vocabulary P_{vocab}, unlike in the original PGN, I first calculate a weighted average of the model's context vector h_t^* and decoder context vector d_t^*, and then feed the resulting vector into a linear layer:

$$P_{vocab} = Softmax(V[h_t^*, d_t^*] + b_{vocab}), \tag{4.6}$$

where V and b_{vocab} are learnable parameters, and P_{vocab} is the probability distribution over all words in the vocabulary. P_{vocab} gives the final probability of predicting the next word w from the vocabulary:

$$P(w) = P_{vocab}(w)\,.$$

Generation probability. The generation probability at timestep t, P_{gen}, is calcu-

lated from the model's context vector h_t^* and the decoder context vector d_t^*:

$$P_{gen} = \delta(w_{h^*}^T h_t^* + w_{d^*}^T d_t^* + b_{pgen}) \,,$$

where w_{h^*}, w_{d^*}, and b_{pgen} are learnable parameters, δ is a sigmoid function, and $P_{gen} \in [0, 1]$.

P_{gen} is used as a soft switch that allows you to choose between generating a word from the vocabulary by sampling from from P_{vocab} (Eq. 4.6) and copying a word from the input sequence by sampling from the attention distribution over the input sequence α^t. This explicitly guides the model on where to look for the next word. I create an extended vocabulary for each batch, which is the union of the vocabulary and all words appearing in the input sentences. As in [See et al., 2017], I obtain the probability distribution over the extended vocabulary as follows:

$$P(w) = P_{gen} P_{vocab} + (1 - P_{gen}) \sum_{i:w_i=w} \alpha_i^t. \tag{4.7}$$

This allows the model to generate out-of-vocabulary (OOV) words, because if w is an OOV word, then P_{vocab} is zero. Similarly, if the word is not in the input batch, then $\sum_{i:w_i=w} \alpha_i^t = 0$.

Fig. 4.2 summarises the workflow in the generator.

4.3.4 Discriminator

Once the tuple has been generated, it is passed through the discriminator along with the input context vector, which classifies the tokens in the tuple as either "real" or "fake", where "real" tokens are those that are in both the generated tuple and the gold tuple, while "fake" tokens are those that are in the generated tuple but not in the gold tuple. The input context vector is a weighted average of the vector representations of the tokens in the input sequence. Thus, the discriminator performs

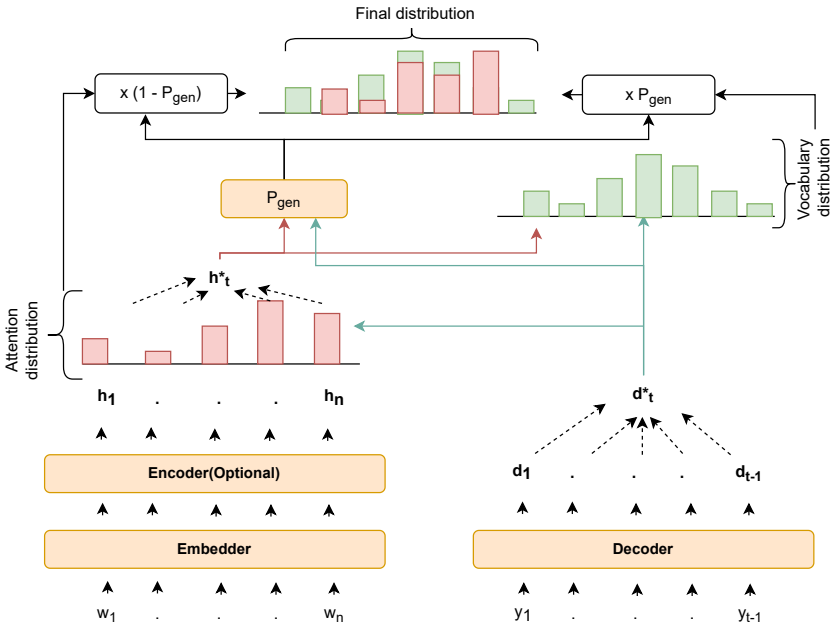


Figure 4.2: The illustration of the workflow in the generator module. Since the encoder is optional, the output of the embedder is used to compute the encoder context vector h_t^*, when it is not used. The decoder context vector d_t^* is computed by attending to the vector representations of the tokens in the entire generated tuple by timestep t. Both the encoder and decoder context vectors are used to compute the generation probability and the distribution over the entire vocabulary as discussed in section 4.3.3.

binary classification of the tokens in the generated tuple while being informed of the input sequence via the input context vector. I consider a discriminator that is composed of a transformer encoder and a sigmoid layer. The transformer encoder computes vector representations of the generated tuple. The vector presentations are then passed through the sigmoid layer for classification as "real" or "fake" as in Eq. 4.2.

Although this approach is similar in its design to generative adversarial networks, the generator in this approach is trained with maximum likelihood rather than adversarially to fool the discriminator.

4.3.5 Training loss

During training, the model loss is the sum of the generator loss and discriminator loss, defined as follows.

Generator loss. I use both the probability of generating the next word $P(w)$ (Eq. 4.7) and the coverage mechanisms to calculate the generator's loss. The loss at timestep t, $loss_t$, is the sum of the negative log likelihood of the target word and the coverage loss:

$$loss_{gen} = -\log P(w_t) + \lambda \sum_i min(\alpha_i^t, c_i^t)\,.$$

The model is penalised by the coverage loss, $\lambda \sum_i min(\alpha_i^t, c_i^t)$, if it repeatedly attends to the same locations. I believe that a specific token from the source sentence can only appear once in a generated fact. As a result, if the final coverage vector is greater or less than one, it is penalised.

Discriminator loss. For the discriminator, compute the negative log-likelihood of

Eq. 4.2:

$$loss_{disc} = -\sum_{i=1}^{m}[(y_i^{real} = y_i)\log(\sigma(w^T[\mathbf{y}_i, \mathbf{d}^*])) + (y_i^{real} \neq y_i)\log(1 - \sigma(w^T[\mathbf{y}_i, \mathbf{d}^*]))].$$

4.4 Experiments, Evaluation, and Results

In this section, I discuss the various experiments that I conducted resulting from the different combinations of modules presented in Section 4.3. I present descriptions of the datasets that I used and the data preparation routines involved. I also discuss the evaluation framework that was used and finally present the results of each experiment in terms of the F_1 and AUC-PR values.

Experiments, training, and evaluation. I set up different combinations of the NN modules in blocks discussed in Section 4.3, and each unique combination resulted in a different model. Thus, I had 15 unique embedder-decoder combinations: (1) BiLSTM - BiLSTM, (2) BiLSTM - Transformer, (3) BiLSTM - Feedback Transfomer, (4) Transformer - BiLSTM, (5) Transformer - Transformer, (6) Transformer - Feedback Transformer, (7) Feedback Transformer - BiLSTM, (8) Feedback Transformer - Transformer, (9) Feedback Transformer - Feedback Transfomer, (10) BERT - BiLSTM, (11) BERT - Transformer, (12) BERT - Feedback Transfomer (13) ELECTRA - BiLSTM, (14) ELECTRA - Transformer, (15) ELECTRA - Feedback Transfomer. I present results for each of these combinations.

I run experiments for all 15 models in six experimental setups: (a) *default* setup with embedders and decoders only; (b) *+ discriminator*, where I add a discriminator to the default setup; (c) *+ transformer encoder*, where I add a transformer-based encoder to the default setup; (d) *+ GNN encoder*, where I add a GNN-based encoder to the default setup; (e) *+ transformer encoder + discriminator*, where I add both a transformer-based encoder and discriminator to the default setup; and (f) *+ GNN*

encoder + discriminator, where I add both a GNN-based encoder and discriminator to the default setup.

For the models with BERT and ELECTRA embedders, I used pre-trained[3] `bert-base-uncased` and `electra-small-discriminator` versions, respectively. Due to resource constraints[4], the parameters of the pretrained models were frozen. All configurations for different module combinations are in Appendix B.3.

All models were trained by minimizing the loss defined in Section 4.3.5. During training, I define a teacher-forcing ratio and randomly choose whether to use teacher forcing or not depending on the value of the randomly generated number compared to the teacher-forcing ratio. The confidence of the extracted fact was calculated by multiplying the probabilities of all tokens in the fact. The checkpoints that were saved after each epoch were then evaluated on the test partition of each dataset using the CaRB [Bhardwaj et al., 2019] evaluation framework. The saved checkpoints did not include the discriminator, as the discriminator was only being used during training.

Datasets. The models were trained and evaluated on three benchmark datasets: OIE2016 [Stanovsky and Dagan, 2016], CaRB [Bhardwaj et al., 2019], and LSOIE [Solawetz and Larson, 2021]. OIE2016 has a total of 5,078 training samples; it is small, which makes it hard to train models that generalize to unseen problem instances. For training purposes, OIE2016 was augmented with samples from another dataset created by [Cui et al., 2018]. This resulted in a huge dataset of more than 36M training samples. This augmented dataset was then trimmed to 1.7M training samples using pre-processing steps presented in Chapter 3, Section 3.4. LSOIE is constructed from the QA-SRL BANK 2.0 [FitzGerald et al., 2018] and contains over 70K sentences and over 150K extraction tuples. CaRB is an improved dataset compared to OIE2016 and is crowdsourced. It was annotated by NLP experts, and it is more accurate than OIE2016 [Bhardwaj et al., 2019]. In addition to binary and explicit extractions,

[3]https://huggingface.co/google/electra-small-discriminator

[4]The models were trained on one GeForce GTX TITAN XP GPU.

CaRB also comes with implicit as well as n-ary extractions. The n-ary extractions were truncated to bear some resemblance to the binary extractions that I considered in this work. For example, in the sentence "A year later, he was appointed Attorney-General for Ireland and on this occasion was sworn of the Privy Council of Ireland", we have the subject "he", relation "was appointed", and objects "Attorney-General", "for Ireland", and "A year later". From this, I formulate the truncated tuple $\langle$*he*, *was appointed*, *Attorney-General for Ireland a year later*$\rangle$.

In each extraction, I introduced special tokens to mark the beginning and end of either subject, relation, or object. "$\langle arg0\rangle$" and "$\langle /arg0\rangle$" were used to indicate the start and end of a subject span, respectively, while "$\langle rel\rangle$" and "$\langle /rel\rangle$" were used to indicate the start and end of a relation span, respectively, and "$\langle arg1\rangle$" and "$\langle /arg1\rangle$" were used to indicate the start and end of an object span, respectively.

In addition to the original versions of the CaRB, OIE2016, and LSOIE datasets, I also generate paraphrased versions, which I use to train our models. I only paraphrase the training sets of the datasets and keep the test sets. I use Parrot[5] to paraphrase samples. Parrot generates paraphrased versions that are adequate and fluent while being as different as possible from the original versions on the surface lexical form. I paraphrased the datasets, so that I get more and diverse data for training our models. Furthermore, as paraphrased samples convey the same meaning as the original samples but with a different lexical form, models trained on this data have a better ability to capture implicit relations.

Results. In Table 4.1, I evaluate our models on CaRB under the experimental setups described in the experiments subsection. It can be noted that in each experimental setup, the model resulting from the combination of an ELECTRA embedder and a feedback transformer decoder outperforms other combinations. The best model on CaRB has an ELECTRA embedder, GNN encoder, feedback transformer, and a

[5]https://github.com/PrithivirajDamodaran/Parrot_Paraphraser

discriminator. This model achieves an F_1 value of 0.747 and an AUC-PR value of 0.740.

Based on the results in Appendix B.1, it can also be noted that the best model on CaRB also achieves the best results on OIE2016. The model achieves an F_1 value of 0.619 and an AUC-PR value of 0.639. Furthermore, the best model achieves an F_1 value of 0.517 and an AUC-PR value of 0.525 on LSOIE. The detailed results and analyses of similar experiments on OIE2016 and LSOIE are in Appendices B.1 and B.2, respectively. Note that I cannot directly compare our results to the previous works by Zhan and Zhao [2020] and Solawetz and Larson [2021] on OIE2016 and LSOIE, respectively, as they use different evaluations. Solawetz and Larson [2021] report the best F_1 and AUC-PR values on LSOIE of only 0.380 and 0.220, respectively.

Results on augmented datasets. On paraphrased and mixed versions of CaRB, I trained the best model from Table 4.1. When training the model on the mixed dataset, each batch was comprised of equal parts of paraphrased and original versions. Following training, the model was tested on the CaRB dataset's original test set. When trained on the paraphrased dataset, the model performs poorly. However, when trained on the mixed dataset, the model outperforms when trained on the original dataset alone. The results are shown in Table 4.2.

Comparison to other systems. In Table 4.3, I compare the best-performing model in this work to earlier OIE systems on CaRB. The results of other systems are directly quoted from earlier publications that use the same experimental parameters and can be directly compared with them. Under the same conditions, I compare the top performing model in this work to the findings provided by Kolluru et al. [2020] (IMoJIE). According to this comparison, the top performing model in this work outperforms the SOTA model, IMoJIE, by 39.63% in F_1 value on CaRB.

		BiLSTM		Transformer		FB Transformer		BERT		ELECTRA	
		F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR
(a)	**BiLSTM**	0.395	0.410	0.442	0.423	0.453	0.477	0.433	0.434	0.445	0.444
	Transformer	0.438	0.389	0.508	0.475	0.599	0.549	0.567	0.553	0.685	0.670
	FB Transformer	0.504	0.400	0.620	0.602	0.706	0.672	0.701	0.673	**0.717**	**0.691**
(b)	**BiLSTM**	0.452	0.420	0.463	0.435	0.474	0.457	0.460	0.458	0.456	0.463
	Transformer	0.632	0.399	0.661	0.482	0.680	0.552	0.675	0.569	0.700	0.696
	FB Transformer	0.541	0.404	0.692	0.676	0.711	0.700	0.710	0.699	**0.726**	**0.719**
(c)	**BiLSTM**	0.410	0.415	0.451	0.435	0.466	0.457	0.467	0.459	0.478	0.463
	Transformer	0.648	0.401	0.669	0.484	0.683	0.562	0.687	0.565	0.699	0.651
	FB Transformer	0.516	0.419	0.701	0.682	0.718	0.685	0.723	0.690	**0.728**	**0.704**
(d)	**BiLSTM**	0.418	0.432	0.465	0.445	0.476	0.469	0.469	0.464	0.468	0.466
	Transformer	0.656	0.412	0.684	0.498	0.703	0.572	0.699	0.568	0.709	0.693
	FB Transformer	0.527	0.423	0.714	0.699	0.728	0.695	0.731	0.698	**0.739**	**0.714**
(e)	**BiLSTM**	0.471	0.439	0.483	0.455	0.485	0.468	0.488	0.480	0.475	0.482
	Transformer	0.457	0.417	0.528	0.482	0.610	0.563	0.614	0.589	0.719	0.715
	FB Transformer	0.560	0.423	0.640	0.622	0.722	0.711	0.737	0.715	**0.745**	**0.738**
(f)	**BiLSTM**	0.473	0.441	0.484	0.456	0.495	0.478	0.489	0.481	0.477	0.484
	Transformer	0.459	0.419	0.529	0.483	0.620	0.573	0.615	0.590	0.721	0.717
	FB Transformer	0.562	0.425	0.641	0.623	0.732	0.721	0.738	0.716	**0.747**	**0.740**

Table 4.1: Results from different module combinations on the CaRB dataset. **FB Transformer** stands for Feedback Transformer. The columns and rows represent embedders and decoders , respectively. We run each module combination across all the experimental setups, (a) to (f), where (a) *default* setup, (b) + *discriminator* setup, (c) + *transformer encoder* setup, (d) + *GNN encoder* setup, (e) + *transformer encoder* + *discriminator* setup, and (f) + *GNN encoder* + *discriminator* setup. The results in bold indicate the best performance in each setup.

4.5 Results Analysis and Ablation Studies

In this section, I will go over the findings reported in Section 4.4. In Section 4.4, I also offer a detailed examination of the contributions of each experimental configuration. In this section, I solely give the analysis of the results from the CaRB dataset. Appendices B.1 and B.2 contain analyses of the results for OIE2016 and LSOEIE, respectively.

Performance across experimental setups. I took into account six experimental configurations, which I defined in Section 4.4. Table 4.5 presents the average performance of all models in each setup on CaRB based on the results presented in Table 4.1.

Dataset	F_1	AUC-PR
Para-phrased	0.509	0.524
Original	0.747	0.740
Mixed	**0.753**	**0.751**

Table 4.2: Performance of the best model from Table 4.1, (ELECTRA + GNN encoder + discriminator), when trained on paraphrased, original, and mixed versions of CaRB.

System	Metric	
	F_1	AUC-PR
Stanford-IE	0.230	0.134
OLLIE	0.411	0.225
OpenIE-4	0.516	0.295
OpenIE-5	0.485	0.257
ClausIE	0.451	0.224
CopyAttention	0.354	0.204
IMoJIE	0.535	0.333
Our best model	**0.747**	**0.740**

Table 4.3: How the best performing model compares to other systems on the CaRB dataset and evaluation framework. The results of previous systems are directly quoted from [Kolluru et al., 2020], because they share the same experimental settings and can be directly compared. Results in bold indicate the best performance.

From the analysis, it can be noted that each experimental setup introduced considerable improvement in performance compared to the control experimental setup, *default*. The setup with the GNN encoder only, *+ GNN encoder*, achieves the best average performance in F_1. It improves on the *default* setup with 11.69%. The setup with both the GNN encoder and discriminator, *+ GNN encoder + discriminator*, achieves the best average performance in AUC-PR. It improves on the *default* setup with 6.53%. Furthermore, I note that models in the *+ GNN encoder* setup perform better than models in the *+ transformer encoder* setup with 1.56% and 2.19% improvements in average F_1 and AUC-PR values, respectively. To that end, I see that the GNN encoder that computes embeddings of the input sequence based on the structure of the dependency tree performs better than the transformer encoder, which does not take into account the structure of the dependency tree when computing embeddings.

Additionally, the best performance is achieved when the GNN encoder is used jointly with the discriminator.

Taking syntactic information into account when computing embeddings not only improves model performance but also improves extraction in some cases. For example, given a sentence »*The twins who came are identical but are uniquely mannered.*«, the best model with the transformer encoder, which does not consider the topology of the dependency tree, yields ⟨*who*, *are*, *uniquely mannered*⟩, whereas the extraction with the GNN encoder, which does consider the topology of the dependency tree, yields ⟨*The twins*, *are*, *uniquely mannered*⟩. This example shows that syntax can be useful in OIE. Thus, I conclude that taking into account the dependency tree's structure when computing embeddings and the discriminative training approach are the main cause of the high boost in the performance of the models presented in this work. This is consistently shown on all three benchmark datasets.

Performance of modules. In addition, I investigated the average performance of the modules in the embedder and decoder blocks. Table 4.6 summarises the analysis based on Table 4.1 results. In terms of F_1 and AUC-PR, models employing a pretrained ELECTRA model get the best average values of 0.653 and 0.625, respectively. Furthermore, models using the feedback transformer decoder attain the best average values in F_1 and AUC-PR, with values of 0.676 and 0.636, respectively. It is self-evident that the feedback transformer performs admirably, and this is largely owing to its ability to capture the input's sequential property, which is critical for tuple generation. However, the other components that are introduced in this chapter boost performance significantly as well. For instance, in F1, the *+GNN encoder* setup achieves the best average performance by 11.69% over the default setup. In AUC-PR, the *+ GNN encoder + discriminator* setup achieves the best average performance by 6.53% over the *default* setup.

Token repetition. In this work, the core tools that control the repetition of tokens

Model	$S \cap T$	Repetition in T
-(*coverage mechanism + gen prob*)	89%	36%
+(*coverage mechanism + gen prob*)	65%	13%

Table 4.4: Results of the best model's ability to introduce new words from the vocabulary into the generated tuple, and to avoid generating repetitive tokens on CaRB. -(*coverage mechanism + gen prob*) is the best model without both *generation probability* and *coverage mechanisms* and +(*coverage mechanism + gen prob*) is the opposite.

during decoding are the *generation probability* and *coverage mechanisms* that I have defined in Section 4.3.3. As such, to check whether these tools are effective, I run the best model in this work with and without the tools and then compute the average percentage of repetitive tokens and the model's ability to introduce new words from the vocabulary. When the best model is run on 50 samples from the test partition with and without both *generation probability* and *coverage mechanisms*. I compute the number of tokens that are both in the source sentence S and the generated tuple T, $S \cap T$, in both settings to measure the model's ability to introduce words from the vocabulary. Similarly, I also compute a percentage of repetitive tokens in T. Table 4.4 summarises the results. From Table 4.4, it can be noted that the percentage of tokens in $|S \cap T|$ drops by 24% when using both *generation probability* and *coverage mechanisms*. This implies that the model copies fewer tokens from the source sentence, and introduces more tokens from the vocabulary, hence the improved ability to generate implicit facts. Furthermore, the repetitive tokens reduce by 23%. To that end, the results confirm that both *generation probability* and *coverage mechanisms* are effective in controlling repetitive tokens, and improving the models' ability to generate implicit facts. This is the first work to look into redundancy at fact level. Sun et al. [2018] and Kolluru et al. [2020] look into redundant information as an issue of generating the same facts multiple times.

Significance of various experimental setups. In addition to the preceding discussion, I investigate whether the improvements brought about by each experimental

configuration in Table 4.1 are substantial rather than a random occurrence. For this, I perform a paired t-test on each new experimental configuration and the default setup, taking into account all paired F_1 and AUC-PR values.

First, the paired t-test produced p values smaller than .05, $p < .05$, in both F_1 and AUC-PR when the *+ transformer encoder* setup was compared to the *default* setup. This suggests that the *default* setup has a less than 5% chance of outperforming the *+ transformer encoder* setup. Second, when the *+ discriminator* and *+ transformer encoder + discriminator* setups are individually matched with the *default* setup, both F_1 and AUC-PR have p values of less than .03, $p < .03$. This suggests that the *default* setup has a less than 3% chance of outperforming either the *+ discriminator* setup or the *+ transformer encoder + discriminator* setup. Finally, when the *+ GNN encoder* and *+ GNN encoder + discriminator* setups are individually paired with the *default* setup, both F_1 and AUC-PR produce p values of less than .01, $p < .01$. This means that the *default* setup has less than a 1% chance of outperforming either the *+ GNN encoder* or *+ GNN encoder + discriminator* setup.

Based on the above considerations, each experimental setup significantly improved the *default* setup in terms of F_1 and AUC-PR compared to the *default* setup. Additionally, the *+ GNN encoder* setup outperforms the *+ transformer encoder* setup. The *+ GNN encoder* setup generates syntactically richer embeddings than the *+ transformer encoder* setup. Furthermore, the *+ GNN encoder + discriminator* setup produces the best results, which can be attributed to the formulation for computing embeddings based on the dependency tree's structure as well as the discriminative training technique. Finally, these patterns may be seen in the results shown in Tables B.1 and B.6, which were derived from the OIE2016 and LSOIE datasets, respectively. These advancements were not coincidental.

Experimental Setup	Avg. F_1	Avg. AUC-PR
Default	0.548	0.522
+ Transformer Encoder	0.603	0.538
+ Discriminator	0.602	0.542
+ GNN Encoder	**0.612**	0.550
+ Transformer Encoder + Discriminator	0.582	0.553
+ GNN Encoder + Discriminator	0.585	**0.556**

Table 4.5: Average performance of all models under different experimental setups on CaRB. Default represents the setup where all models are comprised of just embedders and decoders.

	Module	Avg. F_1	Avg. AUC-PR
(1)	**BiLSTM**	0.506	0.416
	Transformer	0.576	0.525
	BERT	0.612	0.578
	Feedback Transformer	0.615	0.574
	ELECTRA	**0.635**	**0.625**
(2)	**BiLSTM**	0.462	0.452
	Transformer	0.629	0.543
	Feedback Transformer	**0.676**	**0.636**

Table 4.6: Average performance different modules in different blocks for all experiments on CaRB. **(1)** for the embedders, and **(2)** for the decoders. The best average performance for each block is shown in **bold**.

4.6 Related Work

In the recent past, neural OIE models have been developed, and they tackle OIE as either sequence labeling or sequence generation. The former [Stanovsky et al., 2018, Roy et al., 2019, Jiang et al., 2019, Zhan and Zhao, 2020] involves tagging each word in the input text with an appropriate tag that indicates whether the word belongs to either the subject, relation, or object. Methods that approach OIE as sequence generation generate facts one word at a time. These include CopyAttention [Cui et al., 2018], Logician [Sun et al., 2018], and IMoJIE [Kolluru et al., 2020]. CopyAttention is an encoder-decoder model enhanced with copying and attention mechanisms. Logician is another encoder-decoder model that uses coverage attention and gated-dependancy attention to extract facts from Chinese text. IMoJIE outputs

a variable number of different facts per sentence. The next fact from a sentence is best determined in context of all other facts extracted from it so far. Hence, IMoJIE uses a decoding strategy that generates facts in a sequential fashion, one after another, each one being aware of all the ones generated prior to it.

The work presented in this chapter substantially differs from previous OIE works. Firstly, this is the first piece of work to compute embeddings of the input sequences based on the structure of the corresponding dependency trees. This is done by first forming a visibility matrix based on which words are directly related in the dependency tree. Then, a GNN encoder computes syntactically rich embeddings of the input sequences controlled by the formed visibility matrices. This formulation is different from previous approaches where the embeddings of PoS and dependency tags are just concatenated to the embeddings of the according text tokens at embedding level. Secondly, in contrast to all previous works, this work uses an additional module that classifies tokens in the generated tuple as either "real" or "fake" during training. Thirdly, unlike previous neural approaches that tackle OIE as sequence generation, I consider the entire generated sequence by timestep t, $\langle y_1, \ldots, y_{t-1} \rangle$, to compute a decoder context vector, as discussed in Eqs. 4.3–4.4 in Section 4.3.3. Additionally, Sun et al. (2018) and Kolluru et al. (2020) consider redundant information as the issue of generating the same facts multiple times, while I consider redundancy at token level in generated facts. Furthermore, Cui et al. (2018) and Sun et al. (2018) are restrictive in how the models "pick" words when generating facts, which restrains the models' flexibility in generating facts. Cui et al. (2018) only limit the sample space only to the source sentence, and Sun et al. (2018) only consider the source sentence and keywords, while I consider the source sentence and the entire vocabulary.

4.7 Discussion

This work has shown that computing syntactically rich embeddings of text based on the structure of dependency trees significantly improves the performance of neural OIE models. Furthermore, the new method for training OIE models that includes an additional module, the discriminator, which classifies tokens in the generated tuple as either "real" or "fake", boosts the performance of neural OIE models. Additionally, I show that explicitly computing the *generation probability* and the use *coverage mechanisms* substantially improve the models' ability to introduce new words into generated facts and reduce repetitive tokens in generated facts. Finally, I have presented paraphrased versions of OIE2016, CaRB, and LSOIE, and have shown that the models' performance substantially improves when trained on augmented datasets. The best performing model uses both the discriminator and semantically rich embeddings computed by a GNN encoder, and beats the SOTA of IMoJIE on the latest CaRB benchmark dataset. The best model improves the F_1 score of IMoJIE with 39.63%. This model also presents competitive results on other benchmark datasets: OIE2016 and LSOIE.

Chapter 5

Knowledge-Enhanced Open Information Extraction

In this chapter, I build on earlier accomplishments in integrating knowledge from KGs into PLMs to develop a framework for incorporating real-world knowledge into OIE models.

5.1 Introduction

Many recent studies have studied ways on how to incorporate knowledge held in KGs into PLMs, and they have proven to improve PLMs' knowledge content, particularly factual knowledge [Baldini Soares et al., 2019, Zhang et al., 2019, Peters et al., 2019, Logan et al., 2019, Wang et al., 2021]. In most neural OIE techniques, PLMs are the foundational building block that computes contextualised vector representations for input words [Kolluru et al., 2020]. However, no research has been conducted to investigate if enhancing the knowledge content of the base building blocks would improve the performance of neural OIE techniques. This is the first study to investigate how to increase the performance of neural OIE techniques using background knowledge from KGs. To that end, I develop a knowledge-enhanced OIE framework for

both sequence labelling and sequence generation OIE techniques, building on recent advances in incorporating information in PLMs.

Among the ways to incorporate knowledge into PLMs, Zhang et al. [2019] create ERNIE which pre-trains a language representation model on both textual corpora and KGs. They detect entity mentions in text and match them to entities in the full KG. Knowledge embedding algorithms are utilised to generate the embeddings of the respective KG entities, which are subsequently fed into ERNIE. ERNIE integrates entity representations in the knowledge module into the underlying layers of the semantic module based on alignments between text and KGs. Peters et al. [2019] investigate a similar approach with ERNIE and create KnowBert, which includes an integrated entity linker in their model and employs end-to-end training. The entity linker performs entity disambiguation for each potential mention among the candidates. ERNIE and KnowBert both focus solely on entity references and ignore relations.

Logan et al. [2019] observe that language models are only capable of remembering facts that they have seen during training, and these facts are frequently difficult to remember. To solve this, they create a knowledge graph language model that includes methods for detecting and copying information from a knowledge graph that are relevant to the given context. This method enables the model to render previously unseen information as well as construct out-of-vocabulary tokens. Hayashi et al. [2020] present a latent relation language model that uses knowledge graph relations to parameterize the joint distribution of words in a document and the entities that occur in them. This method enhances the models' ability to learn to predict relevant relationships in a given situation. Both Logan et al. [2019] and Hayashi et al. [2020] train superior generation models using relations between entities within a single sentence.

Wang et al. [2021] propose a unified model, KEPLER, for knowledge embed-

ding and pre-trained language representation that incorporates factual knowledge into PLMs and computes effective text-enhanced knowledge embeddings using PLMs. Their method computes entity embeddings from textual descriptions of entities and then jointly optimises the knowledge embedding and language modelling objectives. Their method enhances PLMs' ability to capture factual knowledge.

Previous attempts to incorporate factual knowledge into PLMs have proven to improve PLMs' performance on a variety of NLP tasks, particularly knowledge-driven tasks like entity typing and relation classification [Baldini Soares et al., 2019, Zhang et al., 2019, Peters et al., 2019, Wang et al., 2021]. In the case of KEPLER, this is due to the fact that incorporating knowledge into PLMs improves their ability to extract knowledge from text by requiring the model to encode entities from their corresponding descriptions. This property could also be very useful to OIE, as OIE focuses on extracting knowledge from natural text in the form of triples. In this research, I extend KEPLER, which computes entity representations based on textual descriptions, allowing OIE models to develop more context-aware and knowledgeable embeddings, and propose a knowledge-enhanced framework for OIE.

The main contributions of this chapter include:

- First and foremost, this is the first effort to create a knowledge-enhanced OIE framework based on recent advances in integrating world knowledge from KGs into PLMs. Using this paradigm, I show how world knowledge in PLMs improves the performance of neural OIE models.

- Second, I demonstrate that the KEPLER approach works for PLMs that were not trained using MLM. ELECTRA, which is pre-trained by detecting replaced tokens in the input sequence, was trained by adding a module that predicts masked out tokens. The cross-entropy loss for all masked out positions is used to calculate the MLM loss. ELECTRA was able to gain some world knowledge as a result of this, as well as the supervision from the KG via KE loss.

5.2 My Approach

In this section, I detail the approach that I took to include factual knowledge into PLMs, as well as the knowledge-enhanced framework for OIE. Sections 5.2.1 and 5.2.2 describe how I jointly train KE and MLM objectives on PLMs and explore the knowledge-enhanced OIE framework, respectively.

5.2.1 Fact-aware language representation

I now present the method that I employed to incorporate factual knowledge into linguistic representations. I focus on PLMs that are based on Transformers [Wolf et al., 2020]. As shown in Figure 5.1, I train KE and MLM objectives on the same PLM.

Knowledge representation. I depend on current methodologies for adding factual knowledge into PLMs in this study. I expand on the work of Wang et al. [2021] in a unique approach. I rely on KGs for factual information.

A KG is a graph in which entities are represented as nodes, and relations as edges. As such, a fact in a KG is represented as a triple: $\langle h, r, t \rangle$, where r is the relation, and h and t are head and tail entities, respectively. For each fact in the KG, I retrieve the textual descriptions for h, r, and t using Desc(h), Desc(r), and Desc(t), respectively.

To compute contextualized representations for the textual descriptions for the entities and relations, I use PLMs[1]. Specifically, I consider BERT [Devlin et al., 2019], ALBERT [Lan et al., 2020], and ELECTRA [Clark et al., 2020] for encoding purposes, because these are the PLMs that are used in the OIE models presented in Chapters 3 and 4. Each encoder takes textual descriptions, pre-processed into a sequence of n tokens, $\langle x_1, x_2, \ldots, x_n \rangle$ as input and computes corresponding contextualized representations, $\mathbf{H} \in \mathbb{R}^{n \times d}$, where n is the length of the input sequence, and d is the dimension of each representation.

[1] https://huggingface.co/docs/transformers/index

I extract the embeddings of the first token from the contextualised vector representations of the textual descriptions of the head entity, relation, and tail entity, and use these as embeddings for the actual head entity, relation, and tail entity, respectively. In computing contextualised vector representations, each token is informed of all the other tokens in the sequence, and hence I believe that the embeddings of the first tokens have enough information about the rest of the textual descriptions. I then adopt the translational property [Bordes et al., 2013] for our KE scoring function:

$$d_r(\mathbf{h}, \mathbf{t}) = || \mathbf{h} + \mathbf{r} - \mathbf{t} ||_p ,$$

where I take the norm p as 1, and $\mathbf{h}$, $\mathbf{r}$, and $\mathbf{t}$ are vector representations for the head entity, relation, and tail entity, respectively. The KE loss, LKE, is calculated as:

$$L_{\text{KE}} = -\log \sigma(\gamma - d_r(\mathbf{h}, \mathbf{t})) - \frac{1}{n} \sum_{i=1}^{n} \log \sigma(d_r(\mathbf{h}_i^{'}, \mathbf{t}_i^{'}) - \gamma) ,$$

where $(h^{'}, r, t^{'})$ are negative samples, γ is the margin for margin-based ranking criterion, σ is the sigmoid function, and d_r is the translational property defined above. For a given training batch, I construct negative samples by replacing either the head or tail replaced by a random entity. The KE loss favors higher values of energy for the corrupted samples.

Language representation. For language modelling, I adopt the MLM approach which is used to pre-train most of the language models [Devlin et al., 2019, Lan et al., 2020]. In this training approach, I mask out 15% of tokens in the input sequence and force the model to predict the masked out tokens. For each masked out position, say i, the model computes its corresponding contextualised vector representation, H_i, which is used to predict the token that has been masked out from the entire vocabulary. Unlike other pre-trained language models that are trained through MLM, such as BERT and ALBERT, ELECTRA is pre-trained by detecting replaced tokens in the

input sequence. As such, to fit this into the MLM approach, I add an additional module that takes contextualised vector representations computed by the ELECTRA Model and uses them to predict tokens at masked out positions. Finally, the language modelling loss, L_{MLM}, is calculated as the cross-entropy loss for all the masked out positions.

Joint training loss. During training, I jointly minimize KE and MLM losses:

$$Loss = L_{\text{KE}} + L_{\text{PLM}}.$$

This enables the incorporation of external knowledge from KGs into PLMs. Each training batch, consists of one half of original facts, and another half of corrupted samples. The textual descriptions for the entities and relations in the original facts were used as the input samples for the MLM Encoder.

5.2.2 Knowledge-enhanced open information extraction framework

After training PLMs using the approach discussed in Section 5.2.1, I use the MLM encoder from Figure 5.1 to develop a knowledge-enhanced OIE framework. To develop this framework, I re-visit the approaches developed in Chapters 3 and 4.

Framework for sequence labelling approaches. Sequence labelling OIE approaches use BIO tags [Ratinov and Roth, 2009] to label each token in the input text. Each element of a tuple is implicitly extracted by labeling the according sequence of words with a **B**eginning-tag followed by an arbitrary number of **I**nside-tags, and words that are not part of any extracted phrase are marked with the **O**utside-tag. Tags are prefixed with **A0** if they refer to a tuple's subject, **P** for the predicate, and **A1** to refer to the object of the extracted triple.

In Chapter 3, I present neural models that solve OIE in this manner. Each model

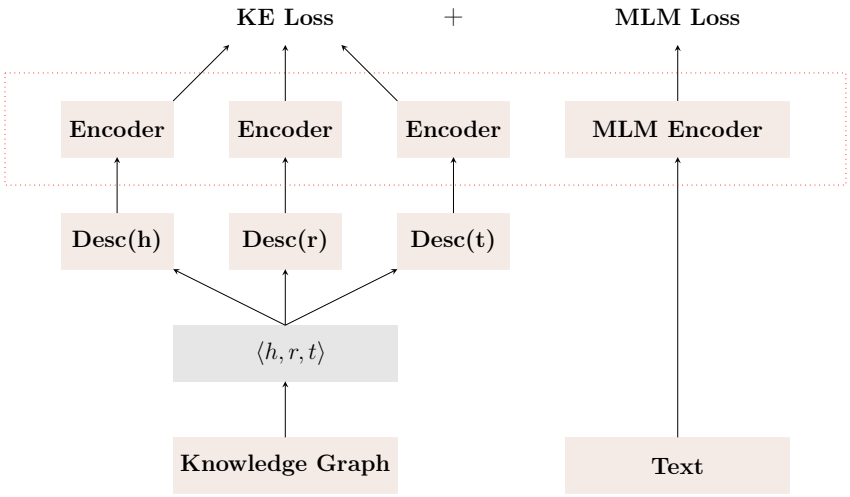


Figure 5.1: The approach for adding factual knowledge to PLMs. I encode entity and relation descriptions as entity and relation embeddings, respectively. I jointly train KE and MLM objectives on the same PLM highlighted in the red dotted rectangle. The encoders, highlighted in the red dotted rectangle, are the components that are shared. $\langle h, r, t \rangle$ represents facts from KG, where h and t are entities, and r is the relation. **Desc(h)**, **Desc(r)**, and **Desc(t)** retrieve descriptions for head entity, relation, and tail entity, respectively.

has three NN blocks: embedding, encoding, and predictor. The embedding block maps text to embedding vectors. The encoding is the middle and optional block of the models that further encodes the embedding vectors. Lastly, the predictor takes the embedding vectors from the encoder and predicts an appropriate BIO-tag for each vector.

To incorporate factual knowledge into sequence labelling approaches, I replace the NN modules in the embedding block with the MLM Encoder from Section 5.2.1. This allows the framework to generate embeddings of the input text using factual-aware embedders. The framework is illustrated in Figure 5.2.

Framework for sequence generation approaches. Sequence generation OIE approaches, given an input sentence, generate tuples one word at a time. In Chap-

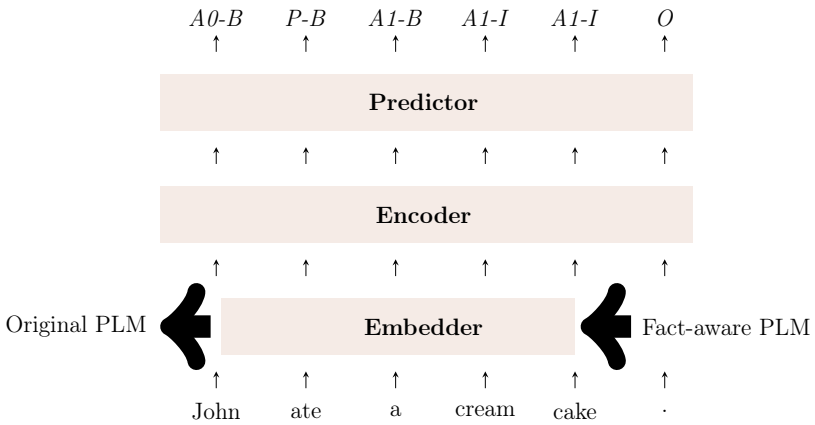


Figure 5.2: The framework for adding factual knowledge to sequence labelling approaches to OIE that are in presented in Chapter 3. I replace the original pre-trained PLM in the embedding block of neural OIE models with a corresponding PLM trained using the approach in Figure 5.1.

ter 4, I present models that solve OIE in this manner. The models have four main blocks: embedding, encoding, decoding, and discriminator blocks. To incorporate factual knowledge into sequence generation approaches, I replace the NN modules in the embedding block with the MLM Encoder from Section 5.2.1. The framework is illustrated in Figure 5.3.

5.3 Experiments, Evaluation, Results, and Analysis

In this section, I discuss the experiments that I conducted to incorporate factual knowledge into PLMs. I also discuss the experiments that were carried out to show the effectiveness of the knowledge-enhanced OIE framework. Additionally, I describe the datasets that were used.

Experiments, training, datasets, and evaluation. The first goal of the experiments in this study is to incorporate factual knowledge into PLMs. To be able to establish that this goal is achieved, I need to evaluate the factual knowledge content of the PLM before, and after training them using the approach defined in Section 5.2.1.

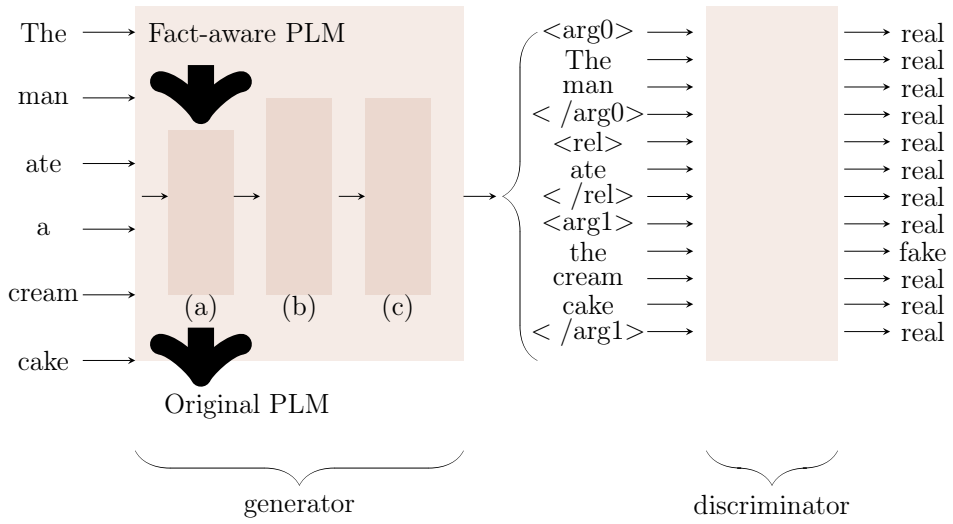


Figure 5.3: The framework for adding factual knowledge to sequence generation OIE approaches that are in presented in Chapter 4. I replace the original pre-trained PLM in the embedding block of the developed neural OIE models with a corresponding PLM trained using the approach in Figure 5.1.

In this work, I consider the base versions[2] of BERT, ALBERT, and ELECTRA language models as baselines. I evaluate the factual content of these baselines using the LAMA dataset [Petroni et al., 2019]. The LAMA dataset is composed of knowledge from various sources that is expressed as facts. These facts are either subject-relation-object triples, or question-answer pairs. In this case, I only focus on the Google-RE[3] part of the LAMA dataset. The Google-RE contains 60K facts that are manually extracted from Wikipedia, and contains five relation types. Specifically, in this work, I only consider two types of relations, namely, "place of birth" and "date of birth", and probe the selected models for this knowledge.

To probe the selected PLMs for "place of birth" or "date of birth" knowledge, each example in the LAMA dataset has the place of birth or date of birth masked out, respectively. Then, the language model is tasked to make the top-k predictions for the masked out tokens. Unlike BERT and ALBERT, ELECTRA is pre-trained by

[2]https://huggingface.co/docs/transformers/index
[3]https://code.google.com/archive/p/relation-extraction-corpus/

detecting replaced tokens in the input sequence. As such, to fit this into this approach, I add an additional module that takes contextualised vector representations computed by the ELECTRA model and uses them to predict tokens at masked out positions. For the top-k predictions, I calculate the weighted hit-rate @k, and take an average of these values for all the samples. The overall average weighted hit-rate @k, Avg. WHR@K, is computed as:

$$\text{Avg. WHR@k} = \frac{1}{n}\sum_{i=1}^{n}\frac{\sum_{j=1}^{k} c_j \times \mathcal{W}_j}{\sum_{j=1}^{k} c_j},$$

where n is the number of samples, k is the number of top predictions being made at that instant, $c_j \in \{0, 1\}$, where 1 indicates that prediction is correct and 0 for wrong predictions, and $\mathcal{W}_j$ is the model's confidence on the prediction. Low values of the Avg. WHR@K imply that the model either gets few answers correct, and with low confidence, while high values indicate the model is getting most answers correct with improved confidence.

After establishing the amount of factual knowledge that is housed in the selected PLMs, I train the PLMs using the approach discussed in Section 5.2.1. For this training, I use the Wikidata5M dataset [Wang et al., 2021]. Wikidata5m is a large-scale KG dataset in which facts from Wikidata [Vrandečić and Krötzsch, 2014] are aligned with textual descriptions from correspond Wikipedia pages. The training approach for incorporating factual knowledge into PLMs requires some negative samples, especially for the KE as discussed in Section 5.2.1. To allow for this, each training batch is comprised of half original samples, and another half of corrupted samples. I corrupt samples by swapping head and tail entities. After training, I extract the MLM encoder and analyse it for the factual knowledge content by computing the overall average weighted hit-rate@k as before. All the results are presented in Table 5.1.

Knowledge-enhanced framework for open information extraction. As dis-

cussed in Section 5.2.2, to incorporate factual knowledge into OIE models that are developed in Chapters 3 and 4, I use the fact-aware MLM Encoder in the embedding block of the developed models. I do not run all the experiments again, I only consider the best models from the two Chapters and replace their embedders with MLM encoders.

For sequence labelling approaches in Chapter 3, the best performing model had an ALBERT embedder, transformer encoder, and an LSTM predictor. The best performance was reached when only optimizing the first and last tokens of the subject, predicate, and object (training scheme d). To assess whether factual knowledge would improve the performance of this model, I replace the original ALBERT embedder with the fact-aware ALBERT MLM encoder. I report the results in Table 5.3

For sequence generation approaches in Chapter 4, the best performing model had an ELECTRA embedder, GAT encoder, feedback transformer decoder and a discriminator. I replace the original ELECTRA embedder with the fact-aware ELECTRA encoder, and I report the results in Table 5.2.

Results. Table 5.1 summarises the findings regarding the incorporation of factual knowledge into PLMs. These findings are the outcome of testing the models both before and after integrating factual knowledge into them. First and foremost, I find that when tasked with predicting the date of birth in both scenarios, the models have low values. Progressing from $k = 1$ to $k = 5$, it can be noted that the values become increasingly lower. Because there is only one valid date of birth, this phenomenon makes sense. However, the models make predictions with a low level of confidence. Another significant finding is that models perform better at predicting locations than they do at guessing dates of birth. Advancing from $k = 1$ to $k = 5$, it can be noted that the values alter slightly. This is due to the models' ability to include synonyms in the top-k predictions. It is worth noting, however, that models struggle with multi-word geographical names. Overall, it can be seen that adding factual knowledge to

		Avg WHR@1	Avg WHR@2	Avg WHR@3	Avg WHR@4	Avg WHR@5
		Date of Birth				
	bert-base-uncased	0.033	0.032	0.030	0.031	0.030
	albert-base-v2	0.034	**0.035**	0.031	0.031	0.031
	electra-small-gen	0.011	0.012	0.011	0.011	0.011
(a)		**Place of Birth**				
	bert-base-uncased	**0.398**	0.364	0.347	0.338	0.331
	albert-base-v2	0.389	0.373	0.362	0.347	0.333
	electra-small-gen	0.282	0.249	0.231	0.219	0.210
		Date of Birth				
	bert-base-uncased	**0.036**	0.034	0.034	0.033	0.032
	albert-base-v2	0.034	**0.036**	0.033	0.034	0.032
	electra-small-gen	0.014	0.014	0.013	0.013	0.014
(b)		**Place of Birth**				
	bert-base-uncased	**0.439**	0.383	0.372	0.371	0.365
	albert-base-v2	0.400	0.384	0.380	0.354	0.353
	electra-small-gen	0.318	0.323	0.301	0.297	0.248

Table 5.1: (a) and (b) present results from evaluating the indicated PLMs on the LAMA dataset before and after incorporating factual knowledge into the PLMs, respectively. The results in bold indicate the best performance.

the models improves their performance.

I then employ the best performing sequence labelling and sequence generation models from Chapters 3 and 4, respectively, to evaluate the effectiveness of the knowledge augmented OIE architecture. The best OIE model for sequence generation was made up of an ELECTRA embedder, a GAT encoder, a feedback transformer decoder, and a discriminator. This model is run with both the original ELECTRA embedder and the fact-aware ELECTRA embedder. The outcomes of these tests are shown in Table 5.2. According to the results, the knowledge augmented OIE framework increases the performance of original sequence generating models, with the best performance being a F_1 score of 0.750 and an AUC-PR of 0.745.

The best sequence labelling OIE model was composed of ALBERT embedder, transformer encoder, and an LSTM predictor. Similarly, I run this model with both the original ALBERT embedder and fact-aware ALBERT embedder, and present the experiment results across the four training schemes in Table 5.3. In brief, the first training scheme, (a), is the natural NLL where all the positions in the sequence

	BiLSTM		Transformer		Feedback Transformer	
	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR
Original ELECTRA	0.477	0.484	0.721	0.717	0.747	0.740
Fact-aware ELECTRA	0.486	0.497	0.732	0.737	**0.750**	**0.745**

Table 5.2: Results of the best sequence generation OIE model with the original ELECTRA and Fact-aware ELECTRA on CaRB Dataset. The rows and columns represent embedders and decoders, respectively. In addition to embedders and decoders, the best model has a GAT encoder and a discriminator as described in Chapter 4. The results in bold indicate the best performance.

	Transformer Encoder + LSTM Predictor							
	(a)		(b)		(c)		(d)	
	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR
Original ALBERT	0.351	0.242	0.555	0.515	0.362	0.278	0.628	0.644
Fact-aware ALBERT	0.365	0.242	0.562	0.529	0.380	0.310	**0.641**	**0.651**

Table 5.3: Results of the best sequence labelling OIE model with the original ALBERT and Fact-aware ALBERT on OIE2016. (a), (b), (c), and (d) are the training schemes introduced in Chapter 3. The results in bold indicate the best performance.

are taken into account when computing the loss. The second training scheme, (b), attempts to decrease the influence of O-tags on the loss term by disregarding the O-tags entirely. The third training scheme, (c), which disregards all probabilities except for those of positions that appear right before or right after a transition. The fourth training scheme, (d), which optimizes only the first and the last position of every element of the extracted triple. From the results, it can be noted that the knowledge-enhanced OIE framework improves the performance of original models across the training schemes. The best performance is achieved in training scheme (d) with an F_1 score and AUC-PR of 0.641 and 0.651, respectively.

5.4 Result Analysis

In this section, I discuss the results presented in Tables 5.1, 5.2, and 5.3.

Incorporating factual knowledge into language models. For the task of adding

	Avg WHR@1	Avg WHR@2	Avg WHR@3	Avg WHR@4	Avg WHR@5
	Date of Birth				
bert-base-uncased	9.61	6.25	12.59	7.28	7.75
albert-base-v2	0.85	2.79	5.30	9.17	3.32
electra-small-gen	26.51	16.59	14.74	13.86	**27.78**
	Place of Birth				
bert-base-uncased	10.47	5.21	7.03	9.83	10.26
albert-base-v2	2.77	2.86	5.18	2.07	5.96
electra-small-gen	12.76	29.64	30.46	**35.77**	17.73

Table 5.4: The performance gains by PLMs after incorporating factual knowledge into them expressed as percentage. The results in bold indicate the best performance.

factual knowledge to PLMs whose results are presented in Table 5.1, it can be noted that my approach improves the amount of knowledge in the PLMs that I considered. For example, when it comes to predicting the date of birth, there is an improvement of 27.78% in the overall average weighted hit-rate at $k = 5$ for the ELECTRA model. Furthermore, note that there is an improvement of 35.77% in the overall average weighted hit-rate at $k = 4$ when predicting places of birth for the ELECTRA model. The rest of the performance gains for the PLMs that I considered are presented as percentages in Table 5.4. To that end, the approach to incorporate factual knowledge to PLMs used in this study achieved a good performance.

Knowledge-enhanced framework for open information extraction. The knowledge-enhanced OIE framework that is proposed in this study introduces substantial performance improvements to both sequence labelling and sequence generation approaches to OIE as seen in the results reported in Tables 5.2 and 5.3. It can be noted that while using this framework, the best sequence generation OIE model improves both in F_1 and AUC-PR values from 0.747 to 0.750 and 0.740 to 0.745, respectively. This represents 0.40% and 0.73% improvements in F_1 and AUC-PR values, respectively. Similarly, the framework improves the performance of my best sequence labelling OIE model both in F_1 and AUC-PR values from 0.628 to 0.641 and 0.644 to 0.651, respectively. This represents 2.07% and 1.09% improvements in

	BiLSTM		Transformer		Feedback Transformer	
	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR
Improvements	1.91	2.64	1.47	2.73	0.40	0.73

Table 5.5: The performance gains of sequence generation OIE approaches when trained using the knowledge enhanced OIE framework expressed as percentage.

	Transformer Encoder + LSTM Predictor							
	(a)		(b)		(c)		(d)	
	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR
Improvements	3.99	2.89	1.26	2.72	4.97	11.51	2.07	1.09

Table 5.6: The performance gains of sequence labelling OIE approaches when trained using the knowledge enhanced OIE framework expressed as percentage. (a), (b), (c), and (d) are the training schemes introduced in Chapter 3.

F_1 and AUC-PR values, respectively. To that end, the proposed framework indeed improves the performance of OIE models. Tables 5.5 and 5.6 present the performance gains due to the proposed framework for sequence generation and sequence labelling OIE approaches, respectively.

Significance of the performance gains. I now assess the significance of the PLMs' improvements in their knowledge of both places and dates of birth in order to prove that the performance gains were truly due to factual knowledge absorbed into the PLM due to the training approach used. This was accomplished by conducting paired t-tests on the same model before and after factual knowledge was included.

First, the paired t-test of BERT before and after adding factual knowledge showed a p value smaller than .01, $p < .01$, on both the dates of birth and places of birth predictions. This implies that BERT without factual knowledge has a less than 1% chance of outperforming BERT with factual knowledge. Second, ALBERT had a p value of less than .03, $p < .03$, implying that there is a 3% possibility that ALBERT without factual information could defeat ALBERT with factual knowledge. Finally, ELECTRA produced a p value of less than .05, $p < .05$, indicating that there is

a less than 5% chance that ELECTRA without factual knowledge can outperform ELECTRA with factual knowledge. The higher value of ELECTRA is most likely due to the fact that the challenge that I chose was based on MLM, whereas ELECTRA is pre-trained by detecting replaced tokens in the input sequence.

Considering the above discussions, each PLM has gained significant knowledge from this approach to incorporating factual knowledge. This increase in factual knowledge is substantial enough to improve the performance of OIE models as shown by the results in Tables 5.5 and 5.6.

5.5 Related Work

Recently, many pieces of work have explored how to incorporate knowledge into PLMs, and they have proved to improve the knowledge content of PLMs, especially factual knowledge. Baldini Soares et al. [2019] take a straightforward "matching the blanks" pre-training objective to help a relation classification task. They take a pair of blank-containing relations that share the same entities and use a training objective that encourages the relation representations to be similar since the pair of relations share the same entities. Their results show that the models using representations learned in this manner significantly outperform previous work on relation extraction.

Zhang et al. [2019] develop ERNIE which pretrains a language representation model on both textual corpora and KGs. They identify entity mentions in text and align them with corresponding entities in the entire KG. The embeddings of the corresponding KG entities are generated using knowledge embedding algorithms and then used as input into ERNIE. Based on the alignments between text and KGs, ERNIE integrates entity representations in the knowledge module into the underlying layers of the semantic module. During training, they mask out some of the named entity alignments in the input text and prompt the model to select appropriate entities

from KGs to complete the alignments. As such, the model learns to aggregate both context and knowledge facts for predicting both tokens and entities. This leads to a knowledgeable language representation model. Peters et al. [2019] explore a similar idea with ERNIE and develop KnowBert which incorporates an integrated entity linker in their model and adopts end-to-end training. The entity linker performs entity disambiguation for each potential mention from among the available candidates.

Logan et al. [2019] note that language models are only capable of remembering facts that they have seen at training time, and often have difficulty recalling these facts. To address this, they develop a knowledge graph language model that is equipped with with mechanisms for identifying and copying facts from a KG that are relevant to the given context. This approach helps the model to render information it has never seen before, as well as generate out-of-vocabulary tokens. Hayashi et al. [2020] introduce a latent relation language models that parameterises the joint distribution over the words in a document and the entities that occur in them through knowledge graph relations. This approach improves the models' ability to learn to predict appropriate relations in a given context. Both [Logan et al., 2019] and [Hayashi et al., 2020] use relations between entities inside one sentence to train better generation models.

In the most recent past, Wang et al. [2021] have developed a unified model for knowledge embedding and pre-trained language representation which integrates factual knowledge into PLMs and computes effective text-enhanced knowledge embedding with strong PLMs. Their approach encodes textual entity descriptions with a PLMs as their embeddings, and then jointly optimises the knowledge embedding and language modelling objectives. The developed approach improves PLMs ability to capture factual knowledge and produces knowledge embeddings that take advantage of the abundant textual information.

In this work, I build on KEPLER to add knowledge to PLMs. Unlike in KEPLER,

the encoders share the same textual descriptions for the entities and relations as their input. This helps the encoders to compute embeddings for text and facts that are close to their context. Additionally, I also use PLMs to encode relation descriptions. After incorporating factual knowledge into PLMs, I use the encoders in the knowledge enhanced OIE framework.

5.6 Conclusion

In this work, I extend the approach by Wang et al. [2021], KEPLER, to add knowledge to PLMs. Unlike KEPLER, the encoders use the same textual descriptions for the entities and relations as their input. This enables encoders to compute embeddings for text and facts that are near to their context. In addition, I use PLMs to encode relation descriptions. With this approach, I am able to add factual knowledge to the PLMs that I considered in this study. After adding factual knowledge into PLMs, I use encoders in the proposed knowledge-enhanced OIE framework. Based on the results of the framework, I show that factual knowledge in PLMs improves the performance of both sequence generation and sequence labelling OIE models.

Chapter 6

Conclusion

I did an extensive study in this dissertation on both sequence tagging and sequence generation neural methods to OIE, motivated by the research goals posed in Chapter 1. This research was motivated by inadequacies in existing neural OIE methodologies and contributes significantly to the field of OIE. In the following sections, I discuss the findings of this research and make recommendations for further work.

6.1 Summary

To begin, I look at addressing the issue of uneven tag frequencies in OIE datasets for sequence tagging methodologies, as suggested by the first research question, **RQ1**. Sequence tagging OIE methods use BIO tags to label each token in the input text. Each element of a tuple is implicitly extracted by labeling the according sequence of words with a **B**eginning-tag followed by an arbitrary number of **I**nside-tags, and words that are not part of any extracted phrase are marked with the **O**utside-tag. Among all BIO tags, the *O*-tag, which indicate tokens that are not part of either the subject, the predicate, or the object, tends to appear at a much higher frequency than the other tags. This, in turn, means that the ability to correctly predict the *O*-tags has the strongest direct influence on the NLL, and encourages models to focus too much

on this label. To address this problem, I develop three novel training schemes for OIE as sequence labelling task. In brief, the first training scheme attempts to decrease the influence of O-tags on the loss term by disregarding the O-tags entirely. The second training scheme disregards all probabilities except for those of positions that appear right before or right after a transition. The third training scheme optimizes only the first and the last position of every element of the extracted triple. From the results presented in Chapter 3, the proposed novel training schemes outperform the natural NLL formulation where all the positions in the sequence are taken into account when computing the loss. Additionally, any task that can be described as a sequence tagging problem can benefit from these novel training approaches.

Second, I look at how to avoid generating redundant tokens in facts and implicit facts in the field of OIE. I explore sequence generation approaches in particular, since they are prone to producing facts that often communicate redundant information. Furthermore, because sequence generation systems generate facts one at a time, they can replicate terms from the whole vocabulary of words. Sun et al. [2018] and Kolluru et al. [2020] look into the issue of redundant information as generating the same fact multiple times. Their approaches are restrictive in how the models "pick" words when generating facts. This restrains the models' flexibility in generating implicit facts. However, sequence generation approaches are also susceptible to generating repetitive text within a fact. I investigate how to make sequence generation OIE techniques less likely to generate repetitive text in facts. I also look into how to increase their ability to generate implicit facts. These lines of research are guided by the research questions **RQ2** and **RQ3**. In Chapter 4, I present mechanisms that control the the generation of repetitive text in facts by explicitly guiding the models to either "pick" words from the vocabulary or the input text. From experiment results, I note that using both *generation probability* and *coverage mechanisms* substantially reduces repetitive tokens in facts. Additionally, I note that when using both *generation probability* and

coverage mechanisms, the models copy fewer tokens from the source sentence, and introduce more tokens from the vocabulary. This implies that the models have an improved ability to generate implicit facts. To that end, the results confirm that both *generation probability* and *coverage mechanisms* are effective in controlling repetitive tokens, and improving the models' ability to generate implicit facts.

Third, I create paraphrased versions of datasets that I use to train the models to increase their capacity to generate facts that are implicitly communicated in text. Only the training sets of the datasets are paraphrased, while the test sets are kept. Paraphrased examples have the same meaning as the original samples but use a different lexical form. To that end, I hypothesise that models trained on this data will be more effective at capturing implicit relationships. From the results presented in Chapter 4, I note that the models perform poorly when trained on the paraphrased dataset. However, when trained on the augmented dataset, the model performs better than when trained on just the original dataset.

Fourth, I should mention that using PoS and dependency tags in addition to the actual input text as a manner of incorporating syntactic information is a typical practise for the task of OIE. The embeddings of the tags are simply concatenated to the embeddings of the associated text tokens in such works where these tags are used. This formulation does not make full advantage of the extensive syntactic information available, particularly that which is found in the structure of dependency trees. In this work, I design a method for maximising the syntactic information provided by the structure of dependency trees guided by our fourth research question, **RQ4**. The developed approach, unlike the previous approaches, computes syntactically rich vector representations of input text tokens guided by the structure of dependency trees. For each input sentence, I construct a visibility matrix of its tokens based on the structure of its dependency tree. I consider tokens that are directly related in the dependency tree as visible to each other. I then feed the token embeddings

and the visibility matrix into a GNN encoder that computes new token embeddings guided by the visibility matrix. This method generates embeddings that improve the performance of OIE models. This is justified by the results presented in Chapter 4.

Fifth, I introduce a new method for training neural OIE models, in addition to computing syntactically rich vector representations. I add an extra module, the discriminator, that takes the generated tuple as input and classifies its tokens as either "real" or "fake", where "real" tokens are those that are in the gold-tuple, while "fake" tokens are those that the generator has missed. Based on the results presented in Chapter 4, the GNN encoder that computes embeddings of the input sequence based on the structure of the dependency tree performs better than the transformer encoder, which does not take into account the structure of the dependency tree when computing embeddings. Additionally, the best performance is achieved when the GNN encoder is used jointly with the discriminator. Thus, I conclude that taking into account the dependency tree's structure when computing embeddings and the discriminative training approach are the main cause of the high boost in the performance of our models. This is consistently shown on all benchmark datasets considered under this work.

Sixth, several studies have looked into ways to combine knowledge stored in KGs into PLMs, and they have shown that doing so improves the knowledge content of PLMs, particularly factual knowledge. In most neural OIE techniques, PLMs are the foundational building element that computes contextualised vector representations for incoming phrases. However, no research has been done to see if increasing the knowledge richness of the fundamental building blocks would improve the performance of neural OIE techniques. The research presented in this dissertation is the first to investigate how to improve the performance of neural OIE techniques using background knowledge from KGs as part of research question **RQ5**. To that end, I build on recent advancements in incorporating knowledge in PLMs and propose a knowledge-

enhanced OIE framework for both sequence tagging and sequence generation OIE approaches. After thorough experiments, I conclude that the knowledge-enhanced OIE framework improves the performance of the models.

6.2 Outlook

Like any other research, the work described in this dissertation has some space for improvement. First, I limited myself to binary relations that are described in a single sentence. However, in the real world, we are constantly confronted with content that spans many sentences and relationships that span multiple sentences. In future research, the concepts and approaches outlined in this dissertation could be expanded to include n-ray relations that span many sentences. Second, models trained on original datasets perform poorly on paraphrased datasets. This is clear from the results shown in Table 4.2. Under normal conditions, one would expect to identify the same sets of facts from all paraphrased versions of the same sentence. This is not true of the OIE models, and the phenomenon shows that models do not understand sentence meaning. Future studies could build on this to look into how OIE models can better capture sentence meanings. Third, I used the formulation for constructing syntactically rich vector representations as well as the new discriminative training approach to only train sequence generation OIE models. It could be examined further in future research whether the proposed methodologies could also be advantageous to OIE sequence labelling approaches. Finally, the majority of OIE model evaluation systems are based on token matching. They do not include the meaning component of each retrieved fact. The test sets do not contain all of the allowed versions of the same facts. As a result, it is difficult to assess the models' performance with confidence. Future research would look into how meaning may be incorporated into evaluation frameworks. This may also lead to a more accurate assessment of implicit facts.

Appendix A

Systematic Comparison of Neural Architectures and Training Approaches for Open Information Extraction

A.1 Analysis with predicate-head hinting

Table A.1 summarizes the results of the experiments with predicate-head hinting, and, as expected, I observed an average improvement of 0.089 F_1 and 0.078 AUC-PR, respectively. Surprisingly, however, the top model in terms of F_1 score in this setting was the one that uses an ALBERT embedding block, a BiLSTM encoding block, and an LSTM prediction block, followed by the model that performed best without predicate-head hinting, which uses a transformer encoding block instead. The latter was once again the top model in terms of AUC-PR, though. In the remainder of this appendix, I have a closer look at these results, which provide a few more interesting insights.

			LSTM				MLP				CRF	Span
			(a)	(b)	(c)	(d)	(a)	(b)	(c)	(d)	(a)	(a)
(1)	**None**	F_1	0.283	0.474	0.390	0.473	0.226	0.325	0.326	0.316	0.307	0.278
		AUC-PR	0.273	0.279	0.246	0.273	0.106	0.106	0.107	0.106	0.110	0.114
	BiLSTM	F_1	0.338	0.663	0.438	0.664	0.166	0.358	0.392	0.628	0.304	0.273
		AUC-PR	0.255	0.699	0.398	**0.710**	0.114	0.302	0.334	**0.602**	0.233	0.155
	BiLSTM-CNN	F_1	0.302	0.542	0.689	0.672	0.164	0.350	0.374	0.627	0.302	0.275
		AUC-PR	0.217	0.579	0.680	0.706	0.100	0.301	0.318	0.590	0.243	**0.116**
	Transformer	F_1	0.339	0.639	0.438	**0.692**	0.142	0.359	0.388	**0.639**	**0.309**	**0.295**
		AUC-PR	0.232	0.642	0.460	0.676	0.069	0.304	0.344	0.599	**0.267**	0.109
(2)	**None**	F_1	0.303	0.502	0.404	0.503	0.235	0.326	0.335	0.325	0.356	0.322
		AUC-PR	0.277	0.274	0.276	0.278	0.111	0.110	0.113	0.114	0.124	0.137
	BiLSTM	F_1	0.333	0.652	0.446	0.700	0.161	0.402	0.395	0.664	0.304	0.317
		AUC-PR	0.250	0.672	0.399	**0.726**	0.120	0.371	0.348	**0.629**	0.243	**0.139**
	BiLSTM-CNN	F_1	0.305	0.552	0.709	0.692	0.173	0.344	0.384	0.636	0.315	0.319
		AUC-PR	0.227	0.599	0.700	0.717	0.130	0.300	0.328	0.593	0.253	0.139
	Transformer	F_1	0.342	0.653	0.457	**0.711**	0.149	0.406	0.398	**0.708**	**0.365**	**0.339**
		AUC-PR	0.233	0.639	0.474	0.724	0.079	0.358	0.348	0.577	**0.287**	0.132

References: Angiras [2018] Stanovsky et al. [2018] Jia and Xiang [2019] Zhan and Zhao [2020]

Table A.1: The results of the evaluation **with** predicate-head hinting, where the different columns correspond with the different training schemes: (a) standard NLL (i.e., considering all labels), (b) disregarding O-tags, (c) optimizing transitions only, and (d) considering start and end of a triple's elements only. The best results are underlined for each of the encoding blocks, and printed boldface for each prediction block. Furthermore, **(1)** represents models trained with a simple embedding block, and **(2)** represents models trained with ALBERT embedding block.

First and foremost, I observed that adding an embedding block to a model led, on average, to a much bigger improvement than this was the case without predicate-head hinting, as illustrated in Figure A.1. More precisely, I found an average improvement of 0.107 F_1, which is about 0.047 F_1 greater than without predicate-head hinting, while the differences among the considered embedding blocks remained about the same. This suggests that all embedding blocks are able to effectively leverage provided details about the predicate head, and, in turn, create better encodings of an input sequence. Furthermore, the transformer encoding block still leads to the greatest average improvement. In contrast to this, using ALBERT instead of a simple

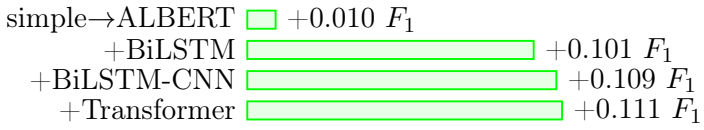


Figure A.1: The average improvements caused by using an ALBERT embedding block in place of a simple one and by adding an encoding block to a model in terms of F_1 score, computed across all experiments performed **with** predicate-head hinting.

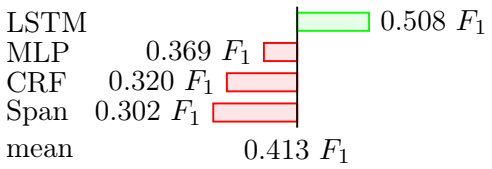


Figure A.2: The average F_1 score achieved **with** predicate-head hinting by each of the different prediction blocks in comparison with the overall mean F_1.

embedding block induced an equally small improvement as before.

Figure A.2 provides a comparison of the different prediction blocks, and, interestingly, the differences among the predictors are smaller with predicate-head hinting. One possible explanation for this is that, in this setting, the encoding block accounts for additional details of an input sequence as part of the computed encoding, which becomes possible given that the predicate head is provided as input to the model. This, in turn, reduces the impact of the employed prediction block. Furthermore, unlike before, the CRF predictor performed, on average, slightly better than the SpanOIE prediction block, whereas LSTM and MLP predictors performed notably better, albeit with a slightly smaller gap between them as without predicate-head hinting.

Finally, Figure A.3 provides a comparison of the different training schemes, and it can be noted that the impact of the same has become bigger than this was the case without predicate-head hinting. This is somewhat surprising, but emphasizes once again that the introduced schemes are more effective than optimizing the standard NLL, and focus on relevant aspects of the considered problem, while paying less attention to incidental details.

Significance of the training schemes. On the findings produced while employing predicate-head hinting presented in Table A.1, I examined the statistical significance of the improvements resulting from training schemes (b), (c), and (d) over the traditional NLL (a). I conducted a paired t-test to examine if there were any differences in the F_1 and AUC-PR values obtained by the models between each of the new schemes and the default NLL. I am only interested in models having either an LSTM or an MLP in the prediction block, as these training schemes can only be employed with these models.

I started with a comparison of the training scheme (b) to the default NLL. The paired t-test on the F_1 and AUC-PR values gave p values of less than .05, $p < .05$, in both cases. According to this research, the default NLL has a less than 5% chance of outperforming the training scheme (b). As a result, the training scheme (b) greatly outperforms the traditional NLL. Second, the paired t-test on the F_1 and AUC-PR values from training schemes (c) and (a) resulted in p values less than .03, $p < .03$. This shows that the default NLL has a slim possibility of outperforming the training scheme (c) by less than 3%. Finally, using the paired t-test on the F_1 and AUC-PR values from training schemes (d) and (a), p values less than .01, $p < .01$,, were obtained. The default NLL has only a 1% chance of outperforming the training scheme (d).

Based on the findings of the above analysis, I can infer that each unique training scheme outperforms the traditional NLL. As a result, I conclude that the gains made by each training scheme are significant rather than random.

A.2 Hyperparameters

Table A.2 summarizes the hyperparameters that were used in the experiments. These were determined over a number of initial experiments, and kept constant throughout

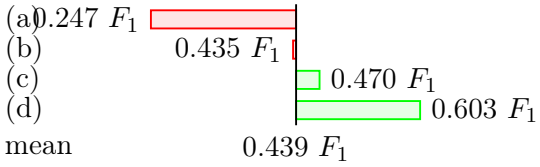


Figure A.3: The average F_1 score achieved **with** predicate-head hinting by each of the different training schemes, computed just for sequence-tagging models in comparison with the overall mean F_1 for the according experiments.

model block	**number of layers**
ALBERT embedder	12
BiLSTM Encoder	2
BiLSTM-CNN Encoder	1
Transformer Encoder	1
MLP Predictor	1
CRF Predictor	1
LSTM Predictor	1
hyperparameter	**value**
batch size	128
dropout rate	0.3
hidden size (except ALBERT)	128
hidden size (ALBERT)	768
learning rate	0.0005
maximum gradient norm	2
weight decay λ	10^{-5}
word-piece embedding size	128
PoS embedding size	100
transformer heads	6
transformer query/key/value size	50
transformer encoder hidden size	512
vocab size	30K

Table A.2: The hyperparameters that were used throughout all the experiments.

all training runs conducted for this research.

Appendix B

Real or Fake? Discriminative Training for Open Information Extraction with Syntactically Rich Embeddings

B.1 OIE2016 Results and Analysis

In this section, I report the findings of my sequence generation models on the OIE2016 dataset. All models were trained and evaluated in all six experimental setups outlined in Section 4.4. I used the CaRB evaluation framework.

Results. Table B.1 displays the results of the model evaluations on the OIE2016 dataset using the experimental setups specified in the experiments subsection. Note that in each experimental configuration, the model produced by combining an ELECTRA embedder and a feedback transformer decoder outperformed other combinations. On the OIE2016 dataset, the best model includes an ELECTRA embedder, a GNN encoder, a feedback transformer, and a discriminator. The model has an F_1

		BiLSTM		Transformer		FB Transformer		BERT		ELECTRA	
		F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR
(a)	**BiLSTM**	0.411	0.438	0.420	0.441	0.421	0.445	0.420	0.446	0.425	0.449
	Transformer	0.439	0.446	0.440	0.447	0.469	0.466	0.471	0.469	0.485	0.471
	FB Transformer	0.472	0.499	0.487	0.498	0.491	0.502	0.494	0.515	0.497	0.521
(b)	**BiLSTM**	0.431	0.450	0.460	0.455	0.467	0.466	0.469	0.468	0.472	0.470
	Transformer	0.497	0.451	0.510	0.488	0.549	0.523	0.560	0.555	0.595	0.602
	FB Transformer	0.546	0.554	0.568	0.572	0.577	0.579	0.599	0.585	0.600	0.626
(c)	**BiLSTM**	0.469	0.468	0.470	0.471	0.485	0.475	0.489	0.477	0.491	0.483
	Transformer	0.504	0.470	0.510	0.476	0.522	0.502	0.562	0.520	0.687	0.660
	FB Transformer	0.531	0.527	0.554	0.536	0.575	0.550	0.599	0.584	0.601	0.607
(d)	**BiLSTM**	0.470	0.471	0.471	0.476	0.488	0.487	0.492	0.489	0.493	0.491
	Transformer	0.528	0.482	0.531	0.489	0.570	0.523	0.581	0.556	0.597	0.604
	FB Transformer	0.548	0.556	0.569	0.576	0.585	0.590	0.594	0.597	0.616	0.633
(e)	**BiLSTM**	0.444	0.463	0.452	0.466	0.472	0.469	0.481	0.477	0.486	0.487
	Transformer	0.491	0.453	0.501	0.472	0.535	0.494	0.551	0.501	0.562	0.567
	FB Transformer	0.541	0.526	0.551	0.540	0.566	0.540	0.582	0.563	0.599	0.592
(f)	**BiLSTM**	0.454	0.472	0.472	0.477	0.479	0.488	0.487	0.485	0.493	0.489
	Transformer	0.521	0.473	0.529	0.481	0.573	0.524	0.581	0.567	0.592	0.607
	FB Transformer	0.568	0.556	0.569	0.579	0.586	0.590	0.602	0.596	**0.619**	**0.639**

Table B.1: Results from different module combinations on the OIE2016 dataset. The columns and rows represent embedders and decoders, respectively. We run each module combination across all the experimental setups, (a) to (f), where (a) *default* setup, (b) *+ discriminator* setup, (c) *+ transformer encoder* setup, (d) *+ GNN encoder* setup, (e) *+ transformer encoder + discriminator* setup, and (f) *+ GNN encoder + discriminator* setup. The results in bold indicate the best performance in each setup.

score of 0.619 and an AUC-PR score of 0.639. I am unable to directly compare these findings to the findings of Zhan and Zhao [2020], because the evaluation framework employed in our study is different.

Performance across experimental setups. Table B.2 presents the average performance of all models in each experimental setup on OIE2016 dataset based on the results presented in Table B.1. Note that each experimental setup introduced considerable improvement in performance compared to the control experimental setup, *default*. The setup with the GNN encoder only, *+ GNN encoder*, achieves the best average performance in F_1. It improves on the *default* setup with 18.90%. The setup with both the GNN encoder and discriminator, *+ GNN encoder + discriminator*,

Experimental Setup	Avg. F_1	Avg. AUC-PR
Default	0.456	0.470
+ Transformer Encoder	0.530	0.514
+ Discriminator	0.527	0.523
+ GNN Encoder	**0.542**	0.533
+ Transformer Encoder + Discriminator	0.521	0.507
+ GNN Encoder + Discriminator	0.542	**0.535**

Table B.2: The average performance of all models under different experimental setups on OIE2016 dataset. Default represents the setup where all models are comprised of just embedders and decoders.

achieves the best average performance in AUC-PR. It improves on the *default* setup with 13.74%. Furthermore, the models in the *+ GNN encoder* setup perform better than models in the *+ transformer encoder* setup with 2.32% and 4.08% improvements in average F_1 and AUC-PR values, respectively. To that end, the GNN encoder which computes embeddings of the input sequence based on the structure of the dependency tree performs better than the transformer encoder which does not take into account the structure of the dependency tree when computing embeddings. Additionally, the best performance is achieved when the GNN encoder is used jointly with the discriminator. Thus, taking into account the dependency tree's structure when computing embeddings and the discriminative training approach are the main cause of the high boost in the performance of OIE models.

Performance of modules. I also looked into the average performance of the modules in the embedder and decoder blocks. Table B.3 summarises the analysis based on results presented in Table B.1. Considering all models, the models with a pretrained ELECTRA model achieve the best average values of 0.545 and 0.550 in F_1 and AUC-PR, respectively. Furthermore, models with the Feedback Transformer decoder achieve the best average values of 0.563 and 0.564 in F_1 and AUC-PR, respectively.

Token repetition. The best model from Table B.1 is run on 50 samples from the test partition of the OIE2016 dataset with and without both *generation probability*

	Module	Avg. F_1	Avg. AUC-PR
(1)	**BiLSTM**	0.492	0.487
	Transformer	0.504	0.496
	BERT	0.534	0.526
	FB Transformer	0.523	0.512
	ELECTRA	**0.545**	**0.550**
(2)	**BiLSTM**	0.464	0.470
	Transformer	0.431	0.508
	FB Transformer	**0.563**	**0.564**

Table B.3: The average performance different modules in different blocks for all experiments on OIE2016 dataset. **FB Transformer** stands for Feedback Transformer. **(1)** for the embedders, and **(2)** for the decoders. The best average performance for each block is indicated in **bold**.

Model	$S \cap T$	Repetition in T
-(*coverage mechanism + gen prob*)	85%	39.50%
+(*coverage mechanism + gen prob*)	67%	15.80%

Table B.4: Results on the best model's ability to introduce new words from the vocabulary into the generated tuple, and to avoid generating repetitive tokens on OIE2016 dataset. -(*coverage mechanism + gen prob*) represents the best model without both *generation probability* and *coverage mechanisms* and +(*coverage mechanism + gen prob*) represents the opposite.

and *coverage mechanisms*. I compute the number of tokens that are both in the source sentence S and the generated tuple T, $S \cap T$, in both settings to measure the model's ability to introduce words from the vocabulary. Similarly, I also compute a percentage of repetitive tokens in T. Table B.4 summarises the results. From Table B.4, the percentage of tokens in $|S \cap T|$ drops by 18% when using both *generation probability* and *coverage mechanisms*. This implies that the model copies fewer tokens from the source sentence, and introduces more tokens from the vocabulary, hence improved ability to generate implicit facts. Further more, the repetitive tokens reduce by 23.70%. To that end, the results confirm that both *generation probability* and *coverage mechanisms* are effective in controlling repetitive tokens, and improving the models' ability to generate implicit facts.

Dataset	F_1	AUC-PR
Para-phrased	0.419	0.449
Original	0.619	0.639
Mixed	**0.701**	**0.699**

Table B.5: The performance of the best model from Table B.1, (ELECTRA + GNN Encoder + Discriminator), when trained on para-phrased, original and mixed versions of the OIE2016 dataset.

Performance on augmented dataset. I trained the best model from Table B.1 on paraphrased and mixed versions of OIE2016 dataset. After training, the model was evaluated on the original test set of OIE2016 dataset. The performance of the model improves when trained on the mixed dataset. Table B.5 summarises the results.

B.2 LSOIE Results and Analysis

In this section, I present results of the models on the LSOIE dataset. I considered all experimental setups discussed in Section 4.4, and I used the CaRB evaluation framework.

Results. In Table B.1, I present the results of evaluating the developed models on the LSOIE dataset under all the experimental setups. In each experimental setup, the model resulting from the combination of an ELECTRA embedder and a Feedback Transformer decoder outperforms other combinations. The best model on LSOIE dataset has an ELECTRA embedder, GNN encoder, Feedback Transformer, and a discriminator. The model achieves an F_1 value of 0.517 and an AUC-PR value of 0.525.

Performance across experimental setups. Table B.7 presents the average performance of all models in each experimental setup on OIE2016 dataset based on the results presented in Table B.6. Each experimental setup introduced considerable improvement in performance compared to the control experimental setup, *default*. The

		BiLSTM		Transformer		FB Transformer		BERT		ELECTRA	
		F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR	F_1	AUC-PR
(a)	**BiLSTM**	0.375	0.381	0.380	0.391	0.391	0.415	0.405	0.421	0.412	0.429
	Transformer	0.428	0.433	0.431	0.438	0.438	0.446	0.448	0.550	0.452	0.457
	FB Transformer	0.456	0.467	0.457	0.472	0.462	0.480	0.475	0.492	0.487	0.501
(b)	**BiLSTM**	0.410	0.397	0.416	0.413	0.429	0.423	0.434	0.431	0.440	0.433
	Transformer	0.428	0.440	0.435	0.448	0.446	0.455	0.451	0.458	0.456	0.462
	FB Transformer	0.435	0.470	0.458	0.476	0.474	0.486	0.480	0.500	0.496	0.519
(c)	**BiLSTM**	0.423	0.399	0.432	0.401	0.435	0.422	0.441	0.434	0.446	0.450
	Transformer	0.433	0.421	0.439	0.435	0.447	0.444	0.463	0.455	0.479	0.471
	FB Transformer	0.459	0.461	0.463	0.467	0.472	0.475	0.490	0.485	0.595	0.504
(d)	**BiLSTM**	0.430	0.413	0.445	0.432	0.439	0.455	0.444	0.454	0.450	0.456
	Transformer	0.438	0.443	0.451	0.456	0.480	0.467	0.493	0.475	0.499	0.485
	FB Transformer	0.469	0.479	0.471	0.499	0.481	0.489	0.499	0.496	0.509	0.524
(e)	**BiLSTM**	0.413	0.401	0.424	0.414	0.437	0.425	0.445	0.439	0.460	0.449
	Transformer	0.427	0.431	0.452	0.446	0.455	0.457	0.459	0.461	0.462	0.463
	FB Transformer	0.465	0.463	0.467	0.468	0.473	0.479	0.499	0.481	0.507	0.502
(f)	**BiLSTM**	0.427	0.415	0.438	0.436	0.449	0.446	0.452	0.449	0.469	0.466
	Transformer	0.439	0.445	0.454	0.458	0.458	0.470	0.463	0.481	0.471	0.486
	FB Transformer	0.471	0.475	0.474	0.492	0.487	0.492	0.502	0.496	**0.517**	**0.525**

Table B.6: Results from different module combinations on the LSOIE dataset. **FB Transformer** stands for Feedback Transformer. The columns and rows represent embedders and decoders , respectively. We run each module combination across all the experimental setups, (a) to (f), where (a) *default* setup, (b) + *discriminator* setup, (c) + *transformer encoder* setup, (d) + *GNN encoder* setup, (e) + *transformer encoder* + *discriminator* setup, and (f) + *GNN encoder* + *discriminator* setup. The results in bold indicate the best performance in each section.

setup with the GNN encoder only, + *GNN encoder*, achieves the best average performance in F_1. It improves on the *default* setup with 7.73%. The setup with both the GNN encoder and discriminator, + *GNN encoder* + *discriminator*, achieves the best average performance in AUC-PR. It improves on the *default* setup with 3.81%. Furthermore, the models in the + *GNN encoder* setup perform better than models in the + *transformer encoder* setup with 1.19% and 4.48% improvements in average F_1 and AUC-PR values, respectively. To that end, the GNN encoder which computes embeddings of the input sequence based on the structure of the dependency tree performs better than the transformer encoder which does not take into account the structure of the dependency tree when computing embeddings. Additionally, the

Experimental Setup	Avg. F_1	Avg. AUC-PR
Default	0.433	0.452
+ Transformer Encoder	0.461	0.448
+ Discriminator	0.446	0.454
+ GNN Encoder	**0.467**	0.468
+ Transformer Encoder + Discriminator	0.456	0.452
+ GNN Encoder + Discriminator	0.465	**0.469**

Table B.7: The average performance of all models under different experimental setups on LSOIE dataset. Default represents the setup where all models are comprised of just embedders and decoders.

	Module	Avg. F_1	Avg. AUC-PR
(1)	**BiLSTM**	0.435	0.435
	Transformer	0.444	0.447
	BERT	0.463	0.470
	FB Transformer	0.453	0.457
	ELECTRA	**0.478**	**0.0.470**
(2)	**BiLSTM**	0.430	0.426
	Transformer	0.452	0.458
	FB Transformer	**0.482**	**0.487**

Table B.8: The average performance different modules in different blocks for all experiments on LSOIE dataset. **(1)** for the embedders, and **(2)** for the decoders. The best average performance for each block is indicated in **bold**.

best performance is achieved when the GNN encoder is used jointly with the discriminator. Thus, taking into account the dependency tree's structure when computing embeddings and the discriminative training approach are the main cause of the high boost in the performance of the developed models.

Performance of modules. I also looked into the average performance of the modules in the embedder and decoder blocks. Table B.8 summarises the analysis based on results presented in Table B.6. Considering all models, the models with a pretrained ELECTRA model achieve the best average values of 0.478 and 0.477 in F_1 and AUC-PR, respectively. Furthermore, models with the Feedback Transformer decoder achieve the best average values of 0.482 and 0.487 in F_1 and AUC-PR, respectively.

Token repetition. The best model from Table B.6 is run on 50 samples from the

Model	$S \cap T$	Repetition in T
-(*coverage mechanism + gen prob*)	82%	34%
+(*coverage mechanism + gen prob*)	68%	11%

Table B.9: Results on the best model's ability to introduce new words from the vocabulary into the generated tuple, and to avoid generating repetitive tokens on LSOIE dataset. -(*coverage mechanism + gen prob*) represents the best model without both *generation probability* and *coverage mechanisms* and +(*coverage mechanism + gen prob*) represents the opposite.

test partition of the LSOIE dataset with and without both *generation probability* and *coverage mechanisms*. I compute the number of tokens that are both in the source sentence S and the generated tuple T, $S \cap T$, in both settings to measure the model's ability to introduce words from the vocabulary. Similarly, I also compute a percentage of repetitive tokens in T. Table B.9 summarises the results. From Table B.9, the percentage of tokens in $|S \cap T|$ drops by 14% when using both *generation probability* and *coverage mechanisms*. This implies that the model copies fewer tokens from the source sentence, and introduces more tokens from the vocabulary, hence improved ability to generate implicit facts. Further more, the repetitive tokens reduce by 23%. To that end, the results confirm that both *generation probability* and *coverage mechanisms* are effective in controlling repetitive tokens, and improving the models' ability to generate implicit facts.

Performance on augmented dataset. I trained the best model from Table B.6 on paraphrased and mixed versions of LSOIE dataset. After training, the model was evaluated on the original test set of LSOIE dataset. The performance of the model improves when trained on the mixed dataset. Table B.10 summarises the results.

B.3 Hyperparameters

Table B.11 summarizes the hyperparameters for all the modules that were used in experiments conducted in this work. These were determined over a number of initial

Dataset	F_1	**AUC-PR**
Para-phrased	0.401	0.424
Original	0.517	0.525
Mixed	**0.610**	**0.605**

Table B.10: The performance of the best model from Table B.6, (ELECTRA + GNN Encoder + Discriminator), when trained on para-phrased, original and mixed versions of the LSOIE dataset.

experiments, and kept constant throughout all training runs conducted for this paper.

model block	**number of layers**
BiLSTM decoder	3
BiLSTM embedder	3
Discriminator	6
ELECTRA embedder	12
Feedback Transformer	6
GAT encoder	3
Transformer	6
hyperparameter	**value**
batch size	128
dropout rate	0.3
dependency tag embedding size	100
embedder/decoder hidden size	256
feedback transformer heads	8
feedback transformer hidden size	512
learning rate	0.0005
maximum gradient norm	2
weight decay λ	10^{-5}
PoS embedding size	100
transformer heads	6
transformer query/key/value size	50
transformer encoder hidden size	512
vocabulary size	30522

Table B.11: The hyperparameters that were used throughout all experiments.

Bibliography

R. Alec, N. Karthik, S. Tim, and I. Sutskever. Improving language understanding by generative pre-training, 2018. Preprint at https://openai.com/blog/language-unsupervised/.

G. Angeli, M. J. Johnson Premkumar, and C. D. Manning. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 344–354. Association for Computational Linguistics, 2015.

A. Angiras. Deep Learning for Open Information Extraction. Master's thesis, University of Oxford, 2018.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

L. Baldini Soares, N. FitzGerald, J. Ling, and T. Kwiatkowski. Matching the blanks: Distributional similarity for relation learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2895–2905. Association for Computational Linguistics, 2019.

M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *Proceedings of the 20th International Joint*

Conference on Artifical Intelligence, IJCAI'07, page 2670–2676. Morgan Kaufmann Publishers Inc., 2007.

S. Bhardwaj, S. Aggarwal, and M. Mausam. CaRB: A crowdsourced benchmark for open IE. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6262–6267. Association for Computational Linguistics, 2019.

A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. *ArXiv*, abs/2003.10555, 2020.

L. D. Corro and R. Gemulla. ClausIE: Clause-Based Open Information Extraction. In *Proceedings of the 22nd International Conference on World Wide Web*, 2013.

L. Cui, F. Wei, and M. Zhou. Neural open information extraction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 407–413. Association for Computational Linguistics, 2018.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

O. Etzioni. Search needs a shake-up. *Nature*, 476(7358):25, 2011.

A. Fader, L. Zettlemoyer, and O. Etzioni. Open question answering over curated and extracted knowledge bases. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1156–1165. ACM, 2014.

A. Fan, T. Lavril, E. Grave, A. Joulin, and S. Sukhbaatar. Accessing higher-level representations in sequential transformers with feedback memory. *ArXiv*, abs/2002.09402, 2020.

N. FitzGerald, J. Michael, L. He, and L. Zettlemoyer. Large-scale QA-SRL parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2051–2060, Melbourne, Australia, 2018. Association for Computational Linguistics.

H. Hayashi, Z. Hu, C. Xiong, and G. Neubig. Latent relation language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:7911–7918, 2020.

L. He, M. Lewis, and L. Zettlemoyer. Question-answer driven semantic role labeling: Using natural language to annotate natural language. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

J. Howard and S. Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339. Association for Computational Linguistics, 2018.

Z. Huang, W. Xu, and K. Yu. Bidirectional lstm-crf models for sequence tagging. *ArXiv*, abs/1508.01991, 2015.

S. Jia and Y. Xiang. Chinese user service intention classification based on hybrid neural network. *Journal of Physics: Conference Series*, 1229, 2019.

Z. Jiang, P. Yin, and G. Neubig. Improving open information extraction via iterative rank-aware learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5295–5300. Association for Computational Linguistics, 2019.

D. Jurafsky and J. H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009. ISBN 0131873210.

K. Kolluru, S. Aggarwal, V. Rathore, Mausam, and S. Chakrabarti. IMoJIE: Iterative memory-based joint open information extraction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5871–5886. Association for Computational Linguistics, 2020.

Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. Albert: A lite bert for self-supervised learning of language representations. In *Proceedings of the International Conference on Learning Representations*, 2020.

R. Logan, N. F. Liu, M. E. Peters, M. Gardner, and S. Singh. Barack's wife hillary: Using knowledge graphs for fact-aware language modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5962–5971. Association for Computational Linguistics, 2019.

M. Mausam. Open information extraction systems and downstream applications. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 2016.

C. Niklaus, M. Cetto, A. Freitas, and S. Handschuh. A survey on open information extraction. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3866–3878. Association for Computational Linguistics, 2018.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word repre-

sentation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 2014.

M. E. Peters, M. Neumann, R. Logan, R. Schwartz, V. Joshi, S. Singh, and N. A. Smith. Knowledge enhanced contextual word representations. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 43–54. Association for Computational Linguistics, 2019.

F. Petroni, T. Rocktäschel, S. Riedel, P. Lewis, A. Bakhtin, Y. Wu, and A. Miller. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, 2019.

L. Ratinov and D. Roth. Design Challenges and Misconceptions in Named Entity Recognition. In *Proceedings of the Annual Conference on Computational Natural Language Learning*, 2009.

A. Roy, Y. Park, T. Lee, and S. Pan. Supervising unsupervised open information extraction models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 728–737. Association for Computational Linguistics, 2019.

A. See, P. J. Liu, and C. D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083. Association for Computational Linguistics, 2017.

S. Soderland, B. Roof, B. Qin, S. Xu, and O. Etzioni. Adapting Open Information Extraction to Domain-specific Relations. *AI Magazine*, 31(3), 2010.

J. Solawetz and S. Larson. LSOIE: A large-scale dataset for supervised open information extraction. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2595–2600. Association for Computational Linguistics, 2021.

G. Stanovsky and I. Dagan. Creating a Large Benchmark for Open Information Extraction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016.

G. Stanovsky, J. Ficler, I. Dagan, and Y. Goldberg. Getting More Out Of Syntax with PropS, 2016. Preprint at http://arxiv.org/abs/1603.01648.

G. Stanovsky, J. Michael, L. Zettlemoyer, and I. Dagan. Supervised open information extraction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 885–895. Association for Computational Linguistics, 2018.

M. Sun, X. Li, X. Wang, M. Fan, Y. Feng, and P. Li. Logician: A unified end-to-end neural approach for open-domain information extraction. In *WSDM '18: Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 556–564, 2018. ISBN 978-1-4503-5581-0.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *Proceedings of the International Conference on Learning Representations*, 2018.

D. Vrandečić and M. Krötzsch. Wikidata: A free collaborative knowledgebase. *Communications of the ACM*, 57:78–85, 09 2014.

X. Wang, T. Gao, Z. Zhu, Z. Zhang, Z. Liu, J. Li, and J. Tang. KEPLER: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 9:176–194, 2021.

T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45. Association for Computational Linguistics, 2020.

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Ł. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144, 2016.

J. Zhan and H. Zhao. Span model for open information extraction on accurate corpus. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:9523–9530, 2020.

Z. Zhang, X. Han, Z. Liu, X. Jiang, M. Sun, and Q. Liu. ERNIE: Enhanced language representation with informative entities. In *Proceedings of the 57th Annual Meeting*

of the Association for Computational Linguistics, pages 1441–1451. Association for Computational Linguistics, 2019.

J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. ISSN 2666-6510.