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

Patrick Hohenecker[1,2,*], **Frank Mtumbuka**[1,*], **Vid Kocijan**[1], **Thomas Lukasiewicz**[1,2]

[1] University of Oxford, Oxford, UK
[2] Serein AI, London, UK

firstname.lastname@cs.ox.ac.uk

Abstract

The goal of open information extraction (OIE) is to extract facts from natural language text, and to represent them as structured triples of the form ⟨*subject*, *predicate*, *object*⟩. For example, given the sentence »*Beethoven composed the Ode to Joy.*«, we are expected to extract the triple ⟨*Beethoven*, *composed*, *Ode to Joy*⟩. In this work, we systematically compare different neural network architectures and training approaches, and improve the performance of the currently best models on the OIE16 benchmark (Stanovsky and Dagan, 2016) by 0.421 F_1 score and 0.420 AUC-PR, respectively, in our experiments (i.e., by more than 200% in both cases). Furthermore, we show that appropriate problem and loss formulations often affect the performance more than the network architecture.

1 Introduction

The field of information extraction (IE) focuses on the automatic acquisition of information from text in a structured format (Jurafsky and Martin, 2009; Niklaus et al., 2018), and methods from this field are used to automatically collect desired information from large corpora of text. Traditionally, IE methods were developed for specific domains with homogeneous corpora and fixed sets of predicates and entities (Niklaus et al., 2018). Such methods are unable to generalize beyond their domains, as they are limited by their predefined collections of entities and predicates.

The area of open information extraction (OIE) was introduced as an alternative approach to IE (Banko et al., 2007), where predicates and entities are not restricted to a specific domain. More formally, the aim of an OIE algorithm is to extract all facts entailed by the input text in the format of ⟨*subject*, *predicate*, *object*⟩ triples. Alternative formulations allow for longer tuples, however, most of the work (including ours) focuses on binary predicates only. Given a sentence »*Sam succeeded in convincing John.*«, an OIE system should extract a tuple: ⟨*Sam*, *succeeded in convincing*, *John*⟩. The relation phrase »*succeeded in convincing*« indicates the semantic relationship between »*Sam*« and »*John*«. OIE plays a key role in several downstream natural language processing (NLP) applications, such as knowledge base construction from text (Soderland et al., 2010), question answering (Fader et al., 2014), information retrieval (Etzioni, 2011), text comprehension, and natural language understanding (Mausam, 2016).

In this work, we focus on the extraction of binary relations, and introduce several novel neural approaches to OIE. We construct our models from three blocks, namely, an embedding block, an encoding block, and a prediction block, and introduce several possible implementations of each of them. Furthermore, we exhaustively test all possible combinations of their use, and show that output encoding and loss function strongly influence the results. To that end, we introduce and test three different training schemes, and investigate their influence on the performance of a model.

All experiments have been conducted on the OIE16 benchmark by Stanovsky and Dagan (2016). As part of the systematic architecture search, several existing neural architectures for OIE (Stanovsky et al., 2018; Zhan and Zhao, 2019; Cui et al., 2018) have been tested and compared under equal conditions.

The main contributions of this work are:

- We introduce and compare different possible training schemes for OIE as a sequence tagging problem.
- We provide a large-scale study of existing and new OIE models, and compare them under the various introduced training schemes.
- We obtain our best results with a novel model based on transformers (Vaswani et al., 2017) and long short-term memories (LSTMs), and

[*] Equal contribution.

Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pages 8554–8565,
November 16–20, 2020. ©2020 Association for Computational Linguistics

improve the current state of the art on the OIE16 benchmark (Stanovsky et al., 2018) by 0.421 F_1 score and 0.420 AUC-PR, respectively, i.e., by more than 200% in both cases.

2 Related Work

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).

Recently, several approaches based on neural network (NN) architectures have been introduced (Stanovsky et al., 2018; Cui et al., 2018; Zhan and Zhao, 2019; Jia and Xiang, 2019). As one of the first neural methods, Stanovsky and Dagan (2016) treat OIE as a sequence-tagging task, and use a bidirectional long short-term memory (BiLSTM) architecture with a softmax layer to predict a BIO tag (Ratinov and Roth, 2009), as defined below, for each word in the input sentence. In contrast to this, Cui et al. (2018) consider OIE as a problem of neural machine-translation (NMT) from English into a triple, and approach it with an attention-based sequence-to-sequence model (Bahdanau et al., 2015). 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, convolutional neural networks (CNNs), and their combinations with conditional random fields (CRFs). Finally, they develop a hybrid model that combines BiLSTMs, CNNs, and CRFs to achieve a maximal performance.

Zhan and Zhao (2019) 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.

3 Task Formulation

As defined earlier, we aim to extract binary relations from single sentences. Each fact 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), we treat OIE as a sequence-tagging task, where each word is labelled with a BIO tag. 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. We use a total of seven different labels, denoted as $\mathcal{L} = \{A0\text{-}B, A0\text{-}I, P\text{-}B, P\text{-}I, A1\text{-}B, A1\text{-}I, O\}$, and Figure 1 illustrates how they are used to extract triples from text.

More recently, Zhan and Zhao (2019) introduced an alternative formulation that is based on spans rather than sequence tags. A span is a subsequence 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 (2019).

Unlike its formal counterpart, semantic parsing, the task of OIE is defined more vaguely, and hence leaves some space for interpretation. For instance, 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***$\rangle$ valid extractions for the first example provided in Figure 1, but ultimately, this is a design choice that depends on the concrete application. Also, 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.

We highlight that single sentences frequently induce more than one triple. As an example of this, consider the lower part of Figure 1, which illustrates a sentence that consolidates two different pieces of information.

Based on the preceding deliberations, we can view

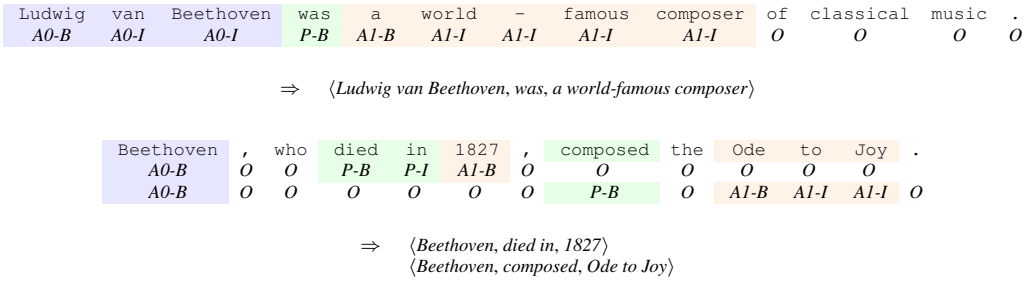

Figure 1: Two examples of the kind of data extraction that is considered in OIE.

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 valid extractions. 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, our aim in this work is to model the conditional distribution $\mathbb{P}\{y \mid X\}$.

4 Methodology

In this work, we consider several variations and extensions of existing architectures for OIE viewed both as sequence-tagging and as span-prediction task. To that end, we subdivide models conceptually into three blocks, which we call embedding, encoding, and prediction (listed from the bottom to the top), and investigate the impact of different NN modules for each of them below. 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 our models, and serves the purpose of mapping text, given as sequence of tokens, to a sequence of embedding vectors. Traditionally, embedding vectors were (pre-)trained 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., 2018; Howard and Ruder, 2018), which induced notable progress on a multitude of NLP tasks.

We consider both approaches, and use either just word-piece embeddings (Wu et al., 2016), which we refer to as simple embedding block, or ALBERT (Lan et al., 2019) for computing contextualized representations.

For the task of OIE, it is a common practice to make use of part-of-speech (PoS) tags in addition to the actual input text (Stanovsky et al., 2018; Jia and Xiang, 2019; Zhan and Zhao, 2019). We 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 was attributed the same PoS tag as the full word it belongs to.

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. In this work, we make use of three different NN modules for encoding embedded sequences: a BiLSTM, the encoder part of a transformer, and a BiLSTM combined with a CNN, as introduced by Jia and Xiang (2019). For the simple BiLSTM encoder, we 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.

Prediction. The top block of a model takes an encoded sequence as input, and computes a probability distribution over all possible triples to extract. We consider four different model architectures as prediction blocks, three for the sequence-tagging setting, based on LSTMs, CRFs, and multi-layer perceptrons (MLPs), and the recently introduced

SpanOIE model (Zhan and Zhao, 2019), which considers OIE as a span-prediction task.

We use LSTMs for predicting label sequences 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.,

$$\mathbb{P}\{\langle y_1, \ldots, y_m\rangle \mid \langle \mathbf{e}_1, \ldots, \mathbf{e}_m\rangle\} = \prod_i \mathbb{P}\{y_i \mid \mathbf{e}_1, \ldots, \mathbf{e}_i, y_1, \ldots, y_{i-1}\}, \quad (1)$$

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 our purposes, we employ a standard linear-chain CRF as prediction block.

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

$$\mathbb{P}\{\langle y_1, \ldots, y_m\rangle \mid \langle \mathbf{e}_1, \ldots, \mathbf{e}_m\rangle\} = \prod_i \mathbb{P}\{y_i \mid \mathbf{e}_i\}, \quad (2)$$

using the same terminology as above.

Finally, we 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. We defer the interested reader to Zhan and Zhao (2019) for any details.

Training Loss. Since prediction blocks aim to model the probability distribution over all candidate triples to extract, the negative log-likelihood (NLL) lends itself as the natural loss function to use. If we consider OIE as a sequence-tagging task, however, then optimizing the NLL requires us to deal with one problematic aspect. Among all BIO tags, the *O*-tag, indicating 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, we explore three novel training schemes, which disregard the probabilities of certain positions in a sequence for computing the NLL, as illustrated in Figure 2. A straightforward attempt to decrease the influence of *O*-tags on the loss term is to disregard them entirely (cf. Figure 2a). Since we model probability distributions over tag sequences as opposed to computing unnormalized 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 2b). 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 2c). Again, remaining tags can be determined from the knowledge of these tags only.

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

5 Experimental Evaluation

Dataset. For our experiments, we 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, we thus augmented the OIE16 training data with samples from

(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*

Figure 2: (a) Excluding all *O* tags when computing the loss. (b) Considering only the tags in the transitions between different tags (c) Considering only the tags in the transitions between different tags that are not *O*.

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, we had to perform a number of preprocessing steps (described in detail in Appendix A, which left us with a total of 1.7M training samples that were combined with the training partition of the OIE16 benchmark dataset.

Experimental Evaluation. We conducted a large-scale study in which we trained and evaluated all combinations of embedding, encoding, and prediction blocks introduced above. For those models containing an ALBERT embedding block, we made use of a pretrained ALBERT model (`albert-base-v1`) provided by Wolf et al. (2019). Due to resource constraints, we had to freeze the parameters of the ALBERT blocks to speed up the training of our models. For models containing a simple embedding block, we used word-piece embeddings (Wu et al., 2016) rather than word embeddings, since these yielded better results in our 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 our 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, 2019; and a slightly modified version the model by Jia and Xiang 2019, as described in Appendix C). We 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).

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 test partition of the OIE16 dataset with respect to both F_1 score and the area under the precision-recall curve (AUC-PR). To that end, we 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, we 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, we 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 a triple ⟨*Donald Trump*, *is*, *president of the U.S.*⟩, then the prediction ⟨*Trump*, *is*, *president of the U.S.*⟩ 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, we run all experiments a second time with predicate-head hinting, and provide the according results in Appendix B.

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

Due to the great number of experiments that have been conducted, it was not possible for us to perform grid search for every one of them separately. Instead, we chose a set of hyperparameters that we 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. Furthermore, the code that was written for our experimental evaluation is available for download on GitHub. [2]

Results. Table 1 summarizes the results of our comparative study, and provides a number of interesting insights. First and foremost, we see 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.

At this point, we want to emphasize that the results that we 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 we decided to not use any kind of pre-selection at all.

[2] https://github.com/phohenecker/emnlp2020-oie

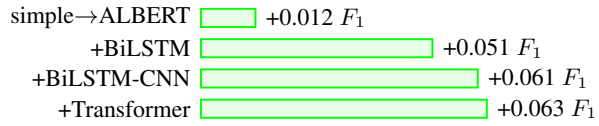

Figure 3: The average improvements caused by using an ALBERT embedding block in place of a simple one and by adding an encoding block to a model in terms of F_1 score, computed across all experiments performed.

Analysis and Ablation Studies. In this section, we further analyze the results presented above, including the ablations that have been performed as part of our comparative study already. Furthermore, we 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 optional. Hence, we 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 summarizes our insights. First and foremost, we 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 performance notably by 0.058 F_1, on average, and, as expected, transformers performed best among all considered encoding blocks.

Next, we compare the various prediction blocks used in our experiments. To that end, Figure 4 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 in the right way, a simple predictor, such as an MLP, can outperform

encoding block	prediction block	LSTM (a)	LSTM (b)	LSTM (c)	LSTM (d)	MLP (a)	MLP (b)	MLP (c)	MLP (d)	CRF (a)	Span (a)
simple embedding block											
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	0.110	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
ALBERT embedding block											
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	0.081	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 (2019)

Table 1: The results of our 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.

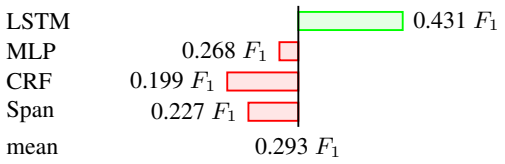

Figure 4: The average F_1 score achieved by each of the different prediction blocks in comparison with the overall mean F_1.

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 for the CRF and the MLP predictor.

Finally, Figure 5 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, we observed significant differences among the training schemes. First and foremost, we see 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.

We 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

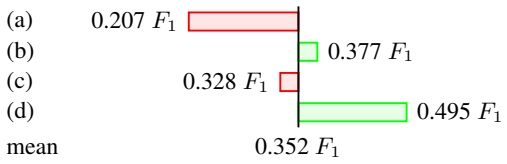

Figure 5: The average F_1 score achieved by each of the different training schemes, computed just for sequence-tagging models in comparison with the overall mean F_1 for the according experiments.

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

to not predict *O*-tags 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 our models, we 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 2 summarizes our 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, we 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 our best-performing model, we 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. To that end, we noted that increasing the number of layers in the transformer encoding block resulted in lower values of both F_1 score and AUC-PR. The same was the case for both increasing and decreasing the used hidden size. Since our 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 our top model seems to address well. Finally, we notice a similar pattern with respect to the different training schemes as the one discussed above.

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: 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.

Some of the existing works on neural OIE employ correction of malformed predicted label sequences (Stanovsky et al., 2018). To that end, orphan intermediate labels, i.e., ones that are preceeded neither by the according head nor by the same intermediate label, are corrected to *O*-tags. In our experiments, this approach did not lead to any improvements, and is thus not further elaborated on. Finally, Appendix B provides additional insights gained in our experiments with predicate-head hinting, which led to an average improvement of 0.074 F_1 score and 0.077 AUC-PR, respectively.

6 Conclusion

In this work, we systematically compared a range of different NN architectures for OIE as well as different schemes for training them. In doing so, we improved the state of the art on the OIE16 benchmark by 0.421 F_1 and 0.420 AUC-PR, respectively, (i.e., by more than 200% in both cases) with a novel model, consisting of an ALBERT embedding block, a transformer encoding block, and an LSTM prediction block, which was trained by means of a training scheme using a newly introduced loss formulation. Subsequent analysis revealed that choos-

ing the right training scheme is as important as selecting the neural model architecture, as the standard NLL loss attributes too much importance to non-essential aspects of the data, and consistently leads to inferior results.

Acknowledgments

Frank Mtumbuka was supported by the Rhodes Trust under a Rhodes Scholarship. This work was also supported by the Alan Turing Institute under the EPSRC grant EP/N510129/1, the AXA Research Fund, the ESRC grant »*Unlocking the Potential of AI for Law*«, and the EPSRC studentship OUCS/EPSRC-NPIF/VK/1123106. We also acknowledge the use of the EPSRC-funded Tier 2 facility JADE (EP/P020275/1) and GPU computing support by Scan Computers International Ltd.

References

Radford Alec, Narasimhan Karthik, Salimans Tim, and Ilya Sutskever. 2018. Improving Language Understanding by Generative Pre-Training. Preprint at https://openai.com/blog/language-unsupervised/.

Gabor Angeli, Melvin Johnson Premkumar, and Christopher D. Manning. 2015. 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 of the Asian Federation of Natural Language Processing*.

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

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations*.

Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. 2007. Open Information Extraction from the Web. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*.

Luciano Del Corro and Rainer Gemulla. 2013. ClausIE: Clause-Based Open Information Extraction. In *Proceedings of the 22nd International Conference on World Wide Web*.

Lei Cui, Furu Wei, and Ming Zhou. 2018. Neural Open Information Extraction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Preprint at http://arxiv.org/abs/1810.04805.

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

Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2014. Open Question Answering over Curated and Extracted Knowledge Bases. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Luheng He, Mike Lewis, and Luke Zettlemoyer. 2015. 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*.

Jeremy Howard and Sebastian Ruder. 2018. Universal Language Model Fine-tuning for Text Classification. Preprint at http://arxiv.org/abs/1801.06146.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF Models for Sequence Tagging. Preprint at http://arxiv.org/abs/1508.01991.

Shengbin Jia and Yang Xiang. 2019. Chinese User Service Intention Classification Based on Hybrid Neural Network. *Journal of Physics: Conference Series*, 1229.

Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing (2nd Edition)*. Prentice-Hall.

Zhen-Zhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. Preprint at http://arxiv.org/abs/1810.04805.

Mausam Mausam. 2016. Open Information Extraction Systems and Downstream Applications. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*.

Christina Niklaus, Matthias Cetto, André Freitas, and Siegfried Handschuh. 2018. A Survey on Open Information Extraction. Preprint at http://arxiv.org/abs/1806.05599.

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

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

Stephen Soderland, Brendan Roof, Bo Qin, Shi Xu, and Oren Etzioni. 2010. Adapting Open Information Extraction to Domain-specific Relations. *AI Magazine*, 31(3).

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

Gabriel Stanovsky, Jessica Ficler, Ido Dagan, and Yoav Goldberg. 2016. Getting More Out Of Syntax with PropS. Preprint at http://arxiv.org/abs/1603.01648.

Gabriel Stanovsky, Julian Michael, Luke Zettlemoyer, and Ido Dagan. 2018. 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)*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. Preprint at http://arxiv.org/abs/1910.03771.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. Preprint at http://arxiv.org/abs/1609.08144.

Junlang Zhan and Hai Zhao. 2019. Span Model for Open Information Extraction on Accurate Corpus. Preprint at http://arxiv.org/abs/1901.10879.

Appendices

A Data Preprocessing

For this work, we 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, we 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), we 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, we thus filtered the dataset with respect to the distribution of predicates in the target triples. We 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, which left us with a dataset consisting of about 28M sentences. Next, we 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. In doing so, we proceeded as follows:

- for predicates appearing less than 500 times, we sampled 50 examples,
- for those with 500 to 1K occurrences, we selected 200 examples,
- predicates with 1K to 10K occurrences were down-sampled to 500 examples,
- for predicates with 10K and 100K occurrences, we sampled 1K examples, and
- for all predicates that appear more than 100K times, we 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. The resulting dataset can be downloaded from our GitHub repository.

B Hyperparameters

Table 4 summarizes the hyperparameters that were used in our experiments. These were determined over a number of initial experiments, and kept constant throughout all training runs conducted for this paper.

C Analysis with Predicate-head Hinting

Table 5 summarizes the results of our experiments with predicate-head hinting, and, as expected, we 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, we have a closer look at these results, which provide a few more interesting insights.

First and foremost, we observed that adding an

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 4: The hyperparameters that were used throughout all our experiments.

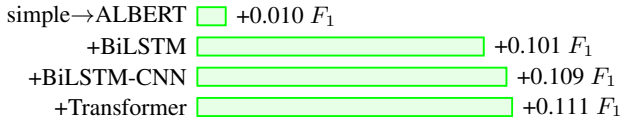

Figure 6: The average improvements caused by using an ALBERT embedding block in place of a simple one and by adding an encoding block to a model in terms of F_1 score, computed across all experiments performed **with** predicate-head hinting.

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 6. More precisely, we 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, we see that the transformer encoding block still leads to the greatest average improvement. In contrast to this, using ALBERT instead of a simple embedding block induced an equally small improvement as before.

Figure 7 provides a comparison of the different prediction blocks, and, interestingly, we see that the differences among the predictors are smaller with predicate-head hinting. One possible explanation for this is that, in this setting, the encoding

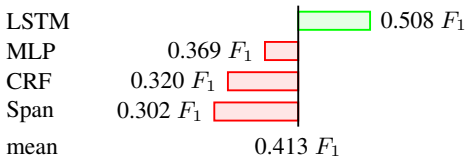

Figure 7: The average F_1 score achieved **with** predicate-head hinting by each of the different prediction blocks in comparison with the overall mean F_1.

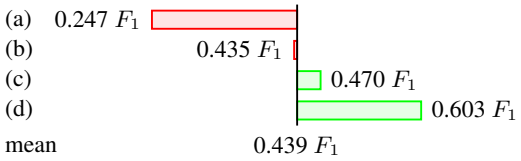

Figure 8: The average F_1 score achieved **with** predicate-head hinting by each of the different training schemes, computed just for sequence-tagging models in comparison with the overall mean F_1 for the according experiments.

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, we observed that, 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 8 provides a comparison of the different training schemes, and we see 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.

D Adjustments of the Model by Jia and Xiang (2019)

As indicated in the main text already, our BiLSTM-CNN encoding block 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

encoding block	prediction block	LSTM (a)	LSTM (b)	LSTM (c)	LSTM (d)	MLP (a)	MLP (b)	MLP (c)	MLP (d)	CRF (a)	Span (a)
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	0.633	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	0.115
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
+ ALBERT embedding block											
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 (2019)

Table 5: The results of our 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.

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, we 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.