Joint Event-Relation Extraction using Encoder Decoder Architecture

Rohit Chowdhury
department of Btech-IT
IIIT Allahabad
UP, India
iit2020043@iiita.ac.in

***Abstract*—We will be using Encoder-Decoder Architecture to extract Joint Event-Relation that describes a method for jointly extracting events and relations from natural language text. The proposed method utilizes an encoder-decoder architecture to address the challenge of extracting multiple events and relations from a single sentence. The first approach uses a sequence labeling model to identify the events and relations in the text, and the second approach employs a pointer network-based model to generate the events and relations jointly.**

***Index Terms*—Encoder-Decoder Architecture, NLP, Sequence labeling, Events and Relation**

INTRODUCTION

The study of natural language processing (NLP) focuses on how computers and human languages interact. Extraction of structured information from unstructured text, such as detecting the event triggers and arguments in a sentence or paragraph, is one of the difficult challenges in NLP. A new NLP application that seeks to solve this issue is joint-event extraction employing encoder-decoder architectures.The use of rule-based systems or machine learning algorithms like support vector machines or random forests is customary for conventional methods of event extraction. However, these techniques are susceptible to the problem of error propagation, in which errors committed in the extraction procedure might have a domino effect and cause additional faults afterwards. Additionally, conventional approaches could miss the intricate connections between event triggers and parameters that exist in natural language text.

On the other hand, encoder-decoder architectures have shown promise in addressing these restrictions. In this method, an encoder processes the input text first to produce a representation of important properties such word embeddings and grammatical structures. The decoder uses this format as input and produces output sequences of event triggers and their related arguments. The decoder captures the dependencies between event triggers and arguments by combining attention mechanisms with recurrent neural networks.Because the model creates output sequences directly rather than generating predictions based on intermediate representations, encoder-decoder designs have the advantage of reducing error propagation. Additionally, by capturing the intricate relationships between event triggers and arguments, this method can enhance accuracy of event extraction.

Overall, joint-event-extraction using encoder-decoder architectures has the potential to improve the accuracy and efficiency of NLP tasks that involve extracting structured information from unstructured text. As the field of NLP continues to evolve, it is likely that we will see further developments in this area that will lead to more accurate and effective approaches to event extraction and other related tasks.

LITERATURE REVIEW

Paper-1

Title: Effective Modeling of Encoder-Decoder Architecture for Joint Entity and Relation Extraction.
Author: Tapas Nayak, Hwee Tou Ng

Summary: The joint extraction of entities and relations from text is a fundamental problem in natural language processing and has numerous applications in information extraction, question answering, and text classification. The proposed encoder-decoder architecture in the paper presents a promising solution to this problem by leveraging the power of deep learning techniques.

The model is designed to encode the input text into a sequence of hidden states, which are then used by the decoder to generate output entities and relations. By jointly optimizing both entity and relation extraction tasks, the model is able to capture the complex interdependencies between them, leading to improved performance on benchmark datasets.

The potential impact of this approach is significant, as it can enable more accurate and efficient processing of natural language data. For example, in information extraction tasks, the model can automatically identify and extract relevant entities and their relationships from large volumes of unstructured text data. This can enable businesses to quickly and accurately extract important insights from customer feedback, news articles, and other sources of unstructured data, leading to more informed decision-making. Overall, the proposed approach has the potential to advance the state-of-the-art in natural language processing and enable new applications in a wide range of fields.

.

Paper-2

Title: Joint Extraction of Entity and Semantic Relation Using Encoder - Decoder Model Based on Attention Mechanism
Author: Yubo Mai, Yatian Shen, Quilin Qi, Xiaxong Shen

Summary: The proposed encoder-decoder model with attention mechanism in the paper presents an innovative solution to the problem of joint extraction of entities and semantic relations from text. The attention mechanism allows the decoder to focus on the relevant parts of the encoded hidden states, improving the accuracy of both entity and relation extraction tasks.

By using a bidirectional LSTM network as the encoder, the model is able to capture the contextual information of the input text, which is essential for accurately identifying entities and relations. The two LSTM networks in the decoder are trained jointly to optimize both tasks simultaneously, leading to improved performance on benchmark datasets.

This approach has the potential to advance natural language processing systems in various applications. For example, it can be used to automatically extract information from medical records to assist healthcare professionals in diagnosis and treatment. It can also be used to analyze social media data to identify influential individuals and their relationships, helping businesses to optimize their marketing strategies.

Overall, the proposed approach in the paper represents a significant advancement in the field of natural language processing and has the potential to enable new applications and improve existing ones.

Paper-3

Title: Joint Entity and Relation Extraction Based on Table Labeling Using Convolutional Neural Networks
Author: Youmi Ma, Tatsuya Hiraoka, Naoaki Okazaki.

Summary: The paper "Joint Entity and Relation Extraction

Based on Table Labeling Using Convolutional Neural Networks" proposes a novel approach for joint entity and relation extraction from tables using Convolutional Neural Networks (CNNs).

The proposed approach uses a table labeling strategy to transform the task of joint entity and relation extraction into a sequence labeling problem. The CNN model takes the table as input and applies convolutional filters to capture local features of the table. The output of the CNN model is then used to label the entities and relations in the table.

The authors evaluate the proposed approach on two benchmark datasets for entity and relation extraction from tables, and compare it against several state-of-the-art methods. The experimental results show that the proposed approach outperforms existing methods in terms of both entity and relation extraction accuracy.

Overall, the paper presents a promising approach to joint entity and relation extraction from tables using CNNs. This approach has the potential to improve the accuracy of natural language processing systems in various applications, such as information extraction, question answering, and text classification, that involve extracting information from tables.

ENCODER-DECODER ARCHITECTURE

In this task, input to the system is a sequence of words, and output is a set of relation tuples. In our first approach, we represent each tuple as entity1 ; entity2 ; relation. We use ';' as a separator token to separate the tuple components. Multiple tuples are separated using the '|' token. We have included one example of such representation in Table 1. Multiple relation tuples with overlapping entities and different lengths of entities can be represented in a simple way using these special tokens (; and |). During inference, after the end of sequence generation, relation tuples can be extracted easily using these special tokens. Due to this uniform representation scheme, where entity tokens, relation tokens, and special tokens are treated similarly, we use a shared vocabulary between the encoder and decoder which includes all of these tokens. The input sentence contains clue words for every relation which can help generate the relation tokens. We use two special tokens so that the model can distinguish between the beginning of a relation tuple and the beginning of a tuple component. To extract the relation tuples from a sentence using the encoder-decoder model, the model has to generate the entity tokens, find relation clue words and map them to the relation tokens, and generate the special tokens at appropriate time. Our experiments show that the encoder decoder models can achieve this quite effectively.

EMBEDDING LAYER & ENCODER

Embedding layers are a type of neural network layer commonly used in natural language processing (NLP) tasks. The goal of an embedding layer is to map each word in a vocabulary to a vector in a continuous embedding space, such that similar words are mapped to nearby points in that space. This vector representation of a word is called its "embedding". Embedding layers are typically learned as part of a larger neural network model, such as a language model or a neural machine translation model. During training, the network learns to adjust the embeddings of each word in the vocabulary to better predict the next word in a sentence or translate a sentence from one language to another. Embedding layers can be pre-trained on large text corpora, allowing the resulting embeddings to capture broad patterns of language use. Alternatively, they can be trained from scratch on smaller datasets, allowing them to capture more specific patterns of language use that are relevant to a particular task.

In the context of natural language processing (NLP), an encoder is a neural network that processes input data (such as text) and produces a fixed-length representation of that input in the form of a vector or sequence of vectors. This fixed-length representation can then be used as input for another neural network, such as a decoder. In the case of text data, the encoder typically consists of one or more layers of recurrent neural network (RNN) cells, such as long short-term memory (LSTM) or gated recurrent unit (GRU) cells. The RNN cells process the input text one token at a time, and each cell produces an output and a hidden state. The hidden state from each cell is then passed as input to the next cell, allowing the encoder to capture information from the entire input sequence. At the end of the encoding process, the final hidden state is typically used as the fixed-length representation of the input text. This hidden state can then be passed to a decoder network, which can use it to generate output text, such as a translation into another language or a response to a question.

We create a single vocabulary V consisting of the source sentence tokens, relation names from relation set R, special separator tokens (';', '|'), start-of-target-sequence token (SOS), end-of-target-sequence token (EOS), and unknown word token (UNK). Word-level embeddings are formed by two components: (1) pre-trained word vectors (2) character embedding-based feature vectors. We use a word embedding layer $E_w \in R^{|V| \times d_w}$ and a character embedding layer $E_c \in R^{|A| \times d_c}$, where d_w is the dimension of word vectors, A is the character alphabet of input sentence tokens, and d_c is the dimension of character embedding vectors. Following Chiu and Nichols (2016), we use a convolutional neural network with max-pooling to extract a feature vector of size d_f for every word. Word embeddings and character embedding based feature vectors are concatenated (k) to obtain the representation of the input tokens.

A source sentence S is represented by vectors of its tokens $x_1, x_2, \ldots, x_n$, where $x_i \in R^{(d_w+d_f)}$ is the vector representation of the ith word and n is the length of S. These vectors x_i are passed to a bi-directional LSTM (Hochreiter and Schmidhuber 1997) (Bi-LSTM) to obtain the hidden representation h^E_i. We set the hidden dimension of the forward and backward LSTM of the Bi-LSTM to be $d_h/2$ to obtain $h^E_i \in R^{d_h}$, where d_h is the hidden dimension of the sequence generator LSTM of the decoder described below.

WORD-LEVEL DECODER & COPY MECHANISM

A decoder model that generates a target sequence represented by word embedding vectors. An LSTM is used as the decoder, which takes the source sentence encoding and the previous target word embedding as input to generate the hidden representation of the current token. The decoder output is projected to the entire vocabulary using a linear layer and bias vector. During training, gold label target tokens are used, but during inference, a masking technique is applied to prevent the decoder from predicting tokens not present in the current sentence, relation set, or special tokens. The UNK token is included in the softmax operation to generate new entities during inference. After decoding is finished, all tuples are extracted based on special tokens, and duplicate and irrelevant tuples are removed. This model is called WordDecoding (WDec).

A target sequence T is represented by only word embedding vectors of its tokens $y_0, y_1, \ldots, y_m$ where $y_i \in R^{d_w}$ is the embedding vector of the ith token and m is the length of the target sequence. y_0 and y_m represent the embedding vector of the SOS and EOS token respectively. The decoder generates one token at a time and stops when EOS is generated. We use an LSTM as the decoder and at time step t, the decoder takes the source sentence encoding (et ∈ R dh) and the previous target word embedding (yt−1) as the input and generates the hidden representation of the current token (h D t ∈ R dh). The sentence encoding vector et can be obtained using the attention mechanism. h D t is projected to the vocabulary V using a linear layer with weight matrix Wv ∈ R |V |×dh and bias vector bv ∈ R |V | (projection layer). h D t = LSTM(etkyt−1, h D t−1) oˆt = Wvh D t + bv, ot = softmax(oˆt) ot represents the normalized scores of all the words in the embedding vocabulary at time step t. h D t−1 is the previous hidden state of the LSTM. The projection layer of the decoder maps the decoder output to the entire vocabulary. During training, we use the gold label target tokens directly. However, during inference, the decoder may predict a token from the vocabulary which is not present in the current sentence or the set of relations or the special tokens. To prevent this, we use a masking technique while applying the softmax operation at the projection layer. We mask (exclude) all words of the vocabulary except the current source sentence tokens, relation tokens, separator tokens (';', '|'), UNK, and EOS tokens in the softmax operation. To mask (exclude) some word from softmax, we set the corresponding value in oˆt at −∞ and the corresponding softmax score will be zero. This ensures the copying of entities from the source sentence only. We include the UNK token in the softmax operation to make sure that the model generates new entities during inference. If the decoder predicts an UNK token, we replace it with the corresponding source word which has the highest attention score. During inference, after decoding is finished, we extract all tuples based on the special tokens, remove duplicate tuples and tuples in which both entities are the same or tuples where the relation token is not from the relation set. This model is referred to as WordDecoding (WDec) henceforth.

POINTERNETWORK-BASED DECODER

The output of the projection layer is utilized in the Pointer Network-based Decoder, a modification of the basic decoder, to point to a particular token in the input sequence or the special tokens. This is helpful for situations where a tiny subset of the input vocabulary serves as the output vocabulary. This method uses two independent linear layers, one for pointing to the input sequence and the other for pointing to the special tokens, to send the decoder output through.

This method uses two independent linear layers, one for pointing to the input sequence and the other for pointing to the special tokens, to send the decoder output through. These two layers' weights are picked up through training. The probability distribution over the input sequence and the special tokens is represented by the output of these layers. At each time step, the decoder generates a probability distribution over the vocabulary and the input sequence using the same process as the Word Decoding approach. The projection layer's output is then used by the pointer network to create two additional probability distributions: one over the input sequence and the other over special tokens. These probability distributions show the likelihood that each token in the input sequence or the special tokens will be pointed to. The vocabulary distribution

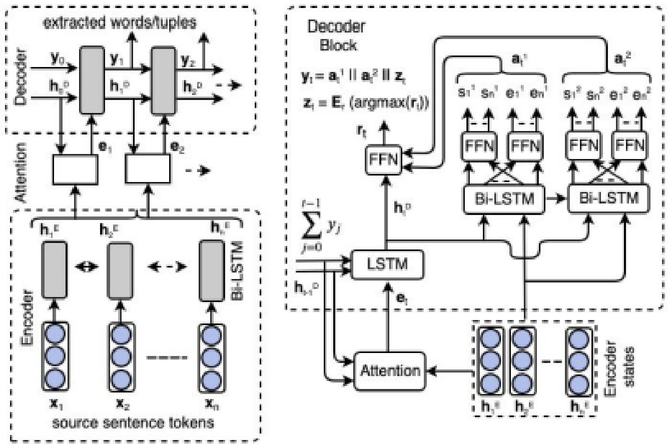

Figure 1: The architecture of an encoder-decoder model (left) and a pointer network-based decoder block (right).

and the two pointer distributions are weighted together to create the final probability distribution that will be used to choose the output token. These distributions' weights are picked up during training. Instead of merely producing new tokens from the output vocabulary, the model can choose tokens from the input sequence or the special tokens thanks to the pointer network-based decoder.

EXPERIMENTAL RESULTS

Among the baselines, HRL achieves significantly higher F1 scores on the two datasets. We run their model and our models five times and report the median results in Table 3. Scores of other baselines in Table 3 are taken from previous published papers (Zeng et al. 2018; Takanobu et al. 2019; Fu, Li, and Ma 2019). Our WordDecoding (WDec) model achieves F1 scores that are 3.9% and 4.1% higher than HRL on the NYT29 and NYT24 datasets respectively. Similarly, our PlaNet Decoding (PNDec) model achieves F1 scores that are 3.0% and 1.3%

higher than HRL on the NYT29 and NYT24 datasets respectively. We perform a statistical significance test (t-test) under a bootstrap pairing between HRL and our models and see that the higher F1 scores achieved by our models are statistically significant ($p < 0.001$). Next, we combine the outputs of five runs of our models and five runs of HRL to build ensemble models. For a test instance, we include those tuples which are extracted in the majority (≥ 3) of the five runs. This ensemble mechanism increases the precision significantly on both datasets with a small improvement in recall as well. In the ensemble scenario, compared to HRL, WDec achieves 4.2% and 3.5% higher F1 scores and PNDec achieves 4.2% and 2.9% higher F1 scores on the NYT29 and NYT24 datasets respectively.

	NYT29			NYT24		
Model	Prec.	Rec.	F1	Prec.	Rec.	F1
Single						
Tagging	0.593	0.381	0.464	0.624	0.317	0.420
CopyR	0.569	0.452	0.504	0.610	0.566	0.587
SPTree	0.492	0.557	0.522	-	-	-
GraphR	-	-	-	0.639	0.600	0.619
HRL	0.692	0.601	0.643	0.781	0.771	0.776
WDec	**0.777**	0.608	**0.682**	**0.881**	0.761	**0.817**
PNDec	0.732	**0.624**	0.673	0.806	**0.773**	0.789
Ensemble						
HRL	0.764	0.604	0.674	0.842	0.778	0.809
WDec	**0.846**	0.621	**0.716**	**0.945**	0.762	**0.844**
PNDec	0.815	**0.639**	**0.716**	0.893	**0.788**	0.838

Table 1: Performance comparison on the two datasets..

ERROR ANALYSIS

The relation tuples extracted by a joint model can be erroneous for multiple reasons such as: (i) extracted entities are wrong; (ii) extracted relations are wrong; (iii) pairings of entities with relations are wrong. To see the effects of the first two reasons, we analyze the performance of HRL and our models on entity generation and relation generation separately. For entity generation, we only consider those entities which are part of some tuple. For relation generation, we only consider the relations of the tuples. We include the performance of our two models and HRL on entity generation and relation generation in Table 5. Our proposed models perform better than HRL on both tasks. Comparing our two models, Ptr Decoding performs better than WordDecoding on both tasks, although WordDecoding achieves higher F1 scores in tuple extraction. This suggests that PtrNetDecoding makes more errors while pairing the entities with relations. We further analyze the outputs of our models and HRL to determine the errors due to ordering of entities (Order), mismatch of the first entity (Ent1), and mismatch of the second entity (Ent2) in Table 6. WordDecoding generates fewer errors than the other two models in all the categories and thus achieves the highest F1 scores on both datasets.

		Model	Prec.	Rec.	F1
NYT29	Ent	HRL	0.833	0.827	0.830
		WDec	**0.865**	0.812	0.838
		PNDec	0.858	**0.851**	**0.855**
	Rel	HRL	0.846	0.745	0.793
		WDec	**0.895**	0.729	0.803
		PNDec	0.884	**0.770**	**0.823**
NYT24	Ent	HRL	0.887	0.892	0.890
		WDec	**0.926**	0.858	0.891
		PNDec	0.906	**0.901**	**0.903**
	Rel	HRL	0.906	0.896	0.901
		WDec	**0.941**	0.880	0.909
		PNDec	0.930	**0.921**	**0.925**

Table 2: Comparison on entity and relation generation tasks.

	NYT29			NYT24		
Model	Order	Ent1	Ent2	Order	Ent1	Ent2
HRL	0.2	5.9	6.6	0.2	4.7	6.3
WDec	0.0	4.2	4.7	0.0	2.4	2.4
PNDec	0.8	5.6	6.0	1.0	4.0	6.1

Table 3: % errors for wrong ordering and entity mismatch.

CONCLUSION

Due to variable entity length, the inclusion of many tuples, and entity overlap across tuples, extracting related tuples from sentences is a difficult operation. In this research, we suggest two original strategies to tackle this task using the encoder-decoder architecture. New state-of-the-art F1 scores are greatly enhanced by our suggested models, according to experiments on the New York Times (NYT) corpus. We would like to investigate our suggested models for a document-level tuple extraction task as future work.

REFERENCES

1. https://link.springer.com/chapter/10.1007/978-3-030-57884-8_54
2. https://ojs.aaai.org/index.php/AAAI/article/view/6374/6230
3. https://aclanthology.org/2022.spnlp-1.2.pdf
4. https://github.com/Rohitciiita/Joint-Event-Relation-Extraction-using-Encoder-Decoder-Architecture
5.