Text Generation from RDF Fact Graphs with Seq2Seq Models

Badr Abdullah, François Buet, Reem Mathbout
University of Lorraine, Nancy

Abstract

In this document, we report the work we have done in order to train (and use) neural seq2seq models to generate text from RDF graphs. Our work is based on the WebNLG dataset which contains paired examples in the form of (RDF triples, text).

1 Introduction

This project aims to build models to generate natural language text from RDF triples. It has been shown in previous work that the neural sequence to sequence (seq2seq) models (also know as encoder-decoder models) can be used for this task. Using this approach, a set of RDF triples can be linearized and represented as a sequence of tokens (source sequence) while the text that verbalizes the triples set is tokenized (and delexilized) to be represented as a sequence of words (target sequence).

A set of RDF triples can be also viewed as a graph. In this project, we investigate whether or not encoding the structure of the graph in the input sequence could improve the text generated by the model.

2 System Description

In this section, we describe the system we developed for this task.

2.1 Software Design

We have decided to re-engineer the software system to make it easier for us to add the functionality that we needed. Our new design is modular and it has been implemented in Python. Different modules hace different roles. For example, the `utils` module contains three sub-modules: (1) `rdf_utils.py` which contains all the procedures we developed in order to parse XML file and manipualte RDF data, (2) which contains all the procedures we developed in order to process text (specially for delexilization) `text_utils.py`, and (3) `sparql_utils.py` which contain the procedure for communicating with DBpedia and retrieving the information we need. We believe that the software design is very flexible and can be easily extended and improved. Figure 1 shows the components in our software design.

Among many procedures, we would like to highlight what we consider the most important procedures in our system:

Semantic type retrieval Each RDF entity can be associated with a semantic type. We retrieve most of the semantic types are retrieved from DBpedia. First, we try to use DBpedia's RANGE class and DOMAIN class associated to the property of each RDF triple (DBpedia can be queried thanks to SPARQL online requests). For instance the range of "nearestCity" is PLACE, and its domain is POPULATEDPLACE. Then, the range class is used as the semantic type of the triple's subject, and the

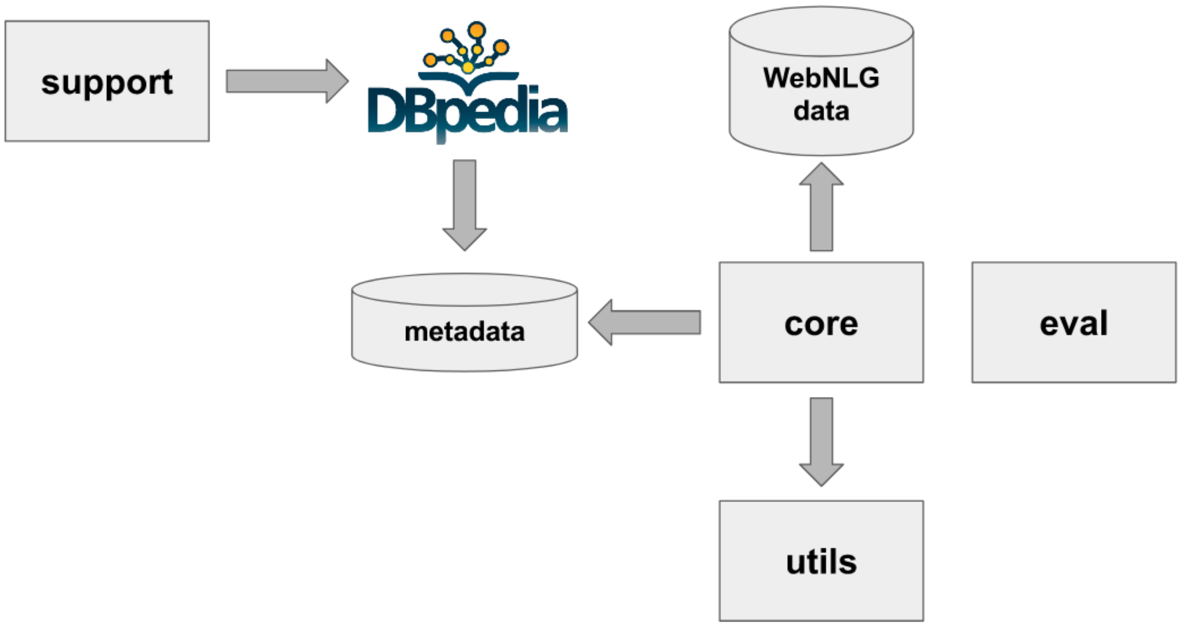


Figure 1: Software system design.

domain class is used as the semantic type of the triple's object. DBpedia does define a range and a range for a significant proportion of the properties that appear in the dataset (40.3%), but it is clearly insufficient to rely only on this method for the semantic typing of all entities. If the search fails, the second method searches for the entity in the resources of DBpedia, and checks if an ontology class is associated with it. Actually, due to the hierarchical structure of DBpedia, several ontology classes are often associated to an entity. For example, "René_Goscinny" is an AGENT, a PERSON, an ARTIST, and a COMICSCREATOR. To decide which of a query's results should be the semantic type of the concerned entity, the frequency of the ontology classes across the dataset is taken into account. More precisely, the entities that correspond to each DBpedia ontology class are counted over the dataset (if the same entity appears multiple times, it is counted each time). To continue the previous example, 11003 entities are identified as AGENT, 9262 as PERSON, 390 as ARTIST, and 326 as COMICSCREATOR. Eventually, the semantic type associated to an entity is the most precise ontology class to appear at least 1,000 times in the dataset (it is parameter that could be tuned, but we only tested this value): the semantic type assigned to "René_Goscinny" is PERSON. The object entities for which no ontology class can be found in DBpedia are given the capitalized name of their corresponding property as semantic type. THING is the semantic type given to subject entities for which no ontology class can be found in DBpedia.

Text delexicalization (To be added)

2.2 Neural Model

For this project, we used the sequence to sequence (seq2seq) model for generating text from RDF triples task. In this model, the two main components are (1) the encoder which takes a sequence as an input and produces a continuous-space vector representation of this sequence, and (2) the decoder which outputs a sequence token-by-token given the representation of the input sequence and the previously predicted words. In the most common architectures, both the encoder and the decoder are recurrent neural networks (RNN) with (gated) cells (either LSTM or GRU). This model is depicted in Figure 1.

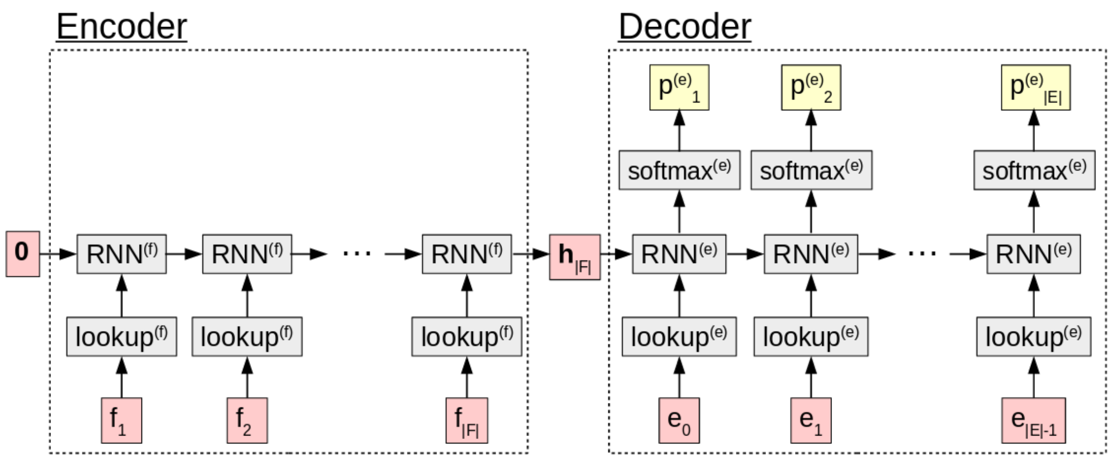


Figure 2: A computation graph of the encoder-decoder model.

In this report, we assume that the reader is familiar with recurrent neural networks. Thus, one could think of the decoder at each time-step as a conditional language model which aims to predict the next word in the output given the previously predicted words as well as the representation of the input sequence. For the task of text generation from RDF triples, we are given an input sequence in a that a representation of an RDF graph F, and the goal is to decode an output sequence in English E by estimating the probability of each word in the output sequence as the conditional probability $P(e_t|e_1^{t-1}, F)$. Thus, if we express the encoder as $RNN^{(f)}(\cdot)$ and the decoder as $RNN^{(e)}(\cdot)$, and if we have a $softmax$ that takes $RNN^{(e)}(\cdot)$'s hidden state at time step t and produces a probability distribution over the vocabulary of English, the Encoder-Decoder can be formally described as follows:

$$\boldsymbol{m}_t^{(f)} = M_{\cdot,f_t}^{(f)}$$

$$\boldsymbol{h}_t^{(f)} = \begin{cases} \text{RNN}^{(f)}(\boldsymbol{m}_t^{(f)}\boldsymbol{h}_{t-1}^{(f)}) & \text{if } t \geq 1 \\ 0 & \text{otherwise.} \end{cases}$$

$$\boldsymbol{m}_t^{(e)} = M_{\cdot,e_{t-1}}^{(e)}$$

$$\boldsymbol{h}_t^{(e)} = \begin{cases} \text{RNN}^{(e)}(\boldsymbol{m}_t^{(e)}\boldsymbol{h}_{t-1}^{(e)}) & \text{if } t \geq 1 \\ \boldsymbol{h}_{|F|}^{(f)} & \text{otherwise.} \end{cases}$$

$$\boldsymbol{p}_t^{(e)} = softmax(W_{hs}\boldsymbol{h}_t^{(e)} + b_s)$$

First, the input sequence F is processed token by token by looking up the embedding of the tth token $\boldsymbol{m}_t^{(f)}$ and updating the encoder's hidden state $\boldsymbol{h}_t^{(f)}$. This hidden state is initialized with $\boldsymbol{h}_0^{(f)} = 0$ and at each time step t, the encoder's hidden state represents an compressed encoding of all the tokens that have been seen in the input sequence. In this model, the last hidden state $\boldsymbol{h}_{|F|}^{(f)}$ is a representation of the entire input sequence . Theoretically, the vector $\boldsymbol{h}_{|F|}^{(e)}$ should preserve all information that have been processed in the input sequence F.

After processing the input sequence, the encoder's last hidden state is used to initialized the decoder's first hidden state and the decoder predicts the probability of the first token in the output sequence E.

Thus, the prediction is conditioned on the input sequence. At each time step t, the embedding of the word e_{t-1} is looked up in order to predict the probability of the word e_t. The hidden state of the decoder is updated in a similar way to the encoder. The only difference is how the decoder is used during training and inference. During training, the ground truth next word is fed as an input to the next time step. During inference, the predicted word (either by sampling or greedy search). The probability $p_t^{(e)}$ is obtained using a $softmax$ on the hidden state $\boldsymbol{h}_t^{(e)}$.

This architecture was first proposed by (Sutskever et al., 2014) and it seems to work very well for moderate-size sequences (tested on machine translation task). However, as the length of the input sequences goes beyond 50 words, this basic architecture does not perform very well. The problem of long-distance dependencies hinders the performance of the system and the fixed-length representation of the input sequence becomes less efficient in preserving the information. To address this drawback, most state-of-the-art implementations of NMT systems includes an attentional decoder. The attention mechanism was introduced by (Bahdanau et al., 2014) and made it easier for the decoder to efficiently translate longer sequences. The main idea is to use a variable-length representation of the input sequence instead of fixed-size vector. Attention-based NMT consists of a bi-directional encoder that processes the input sequence in two directions; forward and backward. Even though we have used attention-based models in our experiments, the details of the attention-based model is beyond the scope of this report. Figure 2 below illustrates the computational graph of a single decoding step given a bi-directional encoder.

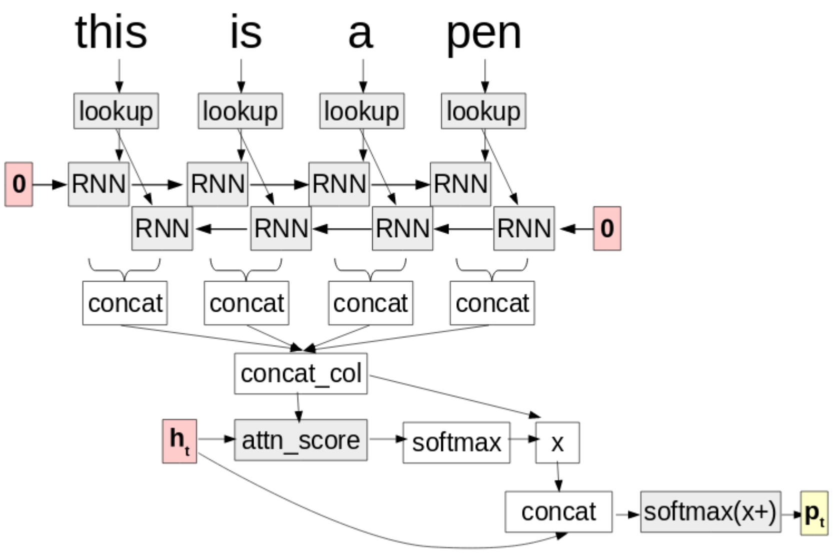


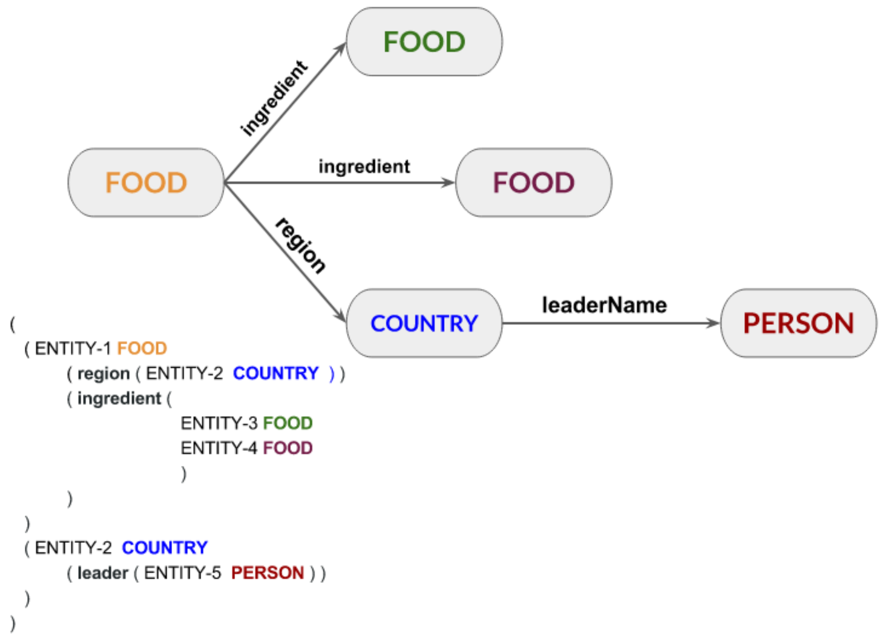
Figure 3: A computation graph for one ”attentional” decoding step. The example in the figure illustrates how the attention-based model is used for machine translation.

3 Implementation

We used the default settings in OpenNMT to build a seq2seq model. That is, a 2-layer attention-based Encoder-Decoder with a hidden layer of size 500. We used dropout with probability 0.3 as a regularization technique. We trained each model for about 26 epochs and selected the model with the best BLEU score on development set.

3.1 Input Representation

We have investigated three different way of representing the RDF triples as a sequence. The first is is simply a linear sequence of the RDF triple set, which we call FLAT sequence. The other two representations are structured representations where we try to encode the structure of the graph in the sequence. Figure 4 shows an example of a graph and a structured sequence representation.



```
(
 ( ENTITY-1 FOOD
      ( region ( ENTITY-2  COUNTRY  ) )
      ( ingredient (
                      ENTITY-3 FOOD
                      ENTITY-4 FOOD
                      )
      )
 )
 ( ENTITY-2  COUNTRY
      ( leader ( ENTITY-5  PERSON ) )
 )
)
```

Figure 4: An example of an RDF graph and its structured representation.

4 Experimental setup

We use the same data as in the WebNLG challenge (Gardent et al., 2017). The results of our developed models are shown in the tables.

Table 1: Result on SEEN data (development set).

Metric	BLUE	METEOR	TER
baseline	*54.03*	*0.39*	*0.40*
modified baseline	*50.03*	*0.39*	*0.43*
FLAT	**48.90**	**0.41**	**0.42**
STR 1	21.57	0.29	0.59
STR 2	28.52	0.32	0.58

5 Conclusion

To be added.

References

Bahdanau, Dzmitry and Cho, Kyunghyun and Bengio, Yoshua 2014. *Neural machine translation by jointly learning to align and translate*. arXiv preprint arXiv:1409.0473

Table 2: Result on UNSEEN data (development set).

Metric	BLUE	METEOR	TER
baseline	*06.13*	*0.07*	*0.80*
modified baseline	*10.59*	*0.14*	*0.88*
FLAT	**22.02**	**0.29**	0.65
STRUCTURED 1	15.69	0.28	**0.60**
STRUCTURED 2	17.30	0.26	0.70

Gardent, Claire, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini 2017. *The WebNLG Challenge: Generating Text from RDF Data*. The WebNLG Challenge: Generating Text from RDF Data.

Sutskever, Ilya, Vinyals, Oriol and Le, Quoc V 2014. *Sequence to sequence learning with neural networks*. Proceedings of Advances in neural information processing systems (NIPS), 2014.