rijksuniversiteit groningen

University of Groningen

Semantic Web Technology

Project Report

Text2Rdf

Authors:
Daniel Bick
Francesca P.
Nivin Pradeep Kumar

November 29, 2020

Contents

1 Introduction **3**

2 Related Work **3**
- 2.1 Information Extraction from Report . . . 4
- 2.2 Bidirectional Encoder Representations from Transformers . . . 4
- 2.3 Recurrent Neural Network for Learning Ontologies . . . 4

3 Methods **4**
- 3.1 Dataset . . . 4
- 3.2 Processing Pipeline . . . 5
- 3.3 Model Architecture . . . 5
- 3.4 Evaluation Procedure . . . 6
  - 3.4.1 Quantitative . . . 6
  - 3.4.2 Qualitative . . . 7

4 Results **7**
- 4.1 Quantitative . . . 8
- 4.2 Qualitative . . . 9

5 Discussion **11**

6 Conclusion **12**

Abstract

In recent years, the amount of available unstructured and semi-structured data has increased by a huge amount. This sort of data represents an undervalued and underutilized trove of information for machine learning applications, like clinical informatics applications, including abnormality tracking systems in the medical field, or semi-automated report writing. The main aim of this research is to convert unstructured text into Resource Description Format (RDF) triples. A machine learning model to automatically convert text documents into ontologies has been developed. The developed model encompasses the pre-trained Bidirectional Encoder Representations from Transformers (BERT) language model as an Encoder, a Soft Attention module, and a Decoder consisting of a Recurrent Neural Network composed of Gated Recurrent Units. Using the developed system, knowledge present in natural language documents is converted into a structured representation that can be easily queried using technologies like SPARQL. Evaluation of the proposed system in terms of precision, recall, and the *F1* scores yield promising results. The code is publicly available on GitHub[1].

1 Introduction

A frequently encountered problem in the area of Machine Learning (ML) is that of converting unstructured or semi-structured to structured information. One area where such systems are pursued is the medical domain, where unstructured or semi-structured radiology reports could be converted into structured forms using ML, such that findings reported in these reports could be used for training other ML classifiers on detecting diseases or suggesting treatments given the structured findings extracted by the former ML model [1]. Another line of research approached the problem of integrating unstructured or semi-structured information from different websites and sources to construct a single, more structured website on which citizens could get all information they were potentially interested in with respect to continued education [2].

One question that naturally arises when considering the transformation from unstructured to structured data is how to represent the structured data. One option for doing so is by expressing extracted information in *Resource Description Format*, i.e. by converting it into RDF triples [3], where each triple contains one piece of information extracted from an input sentence. Having this idea in mind, one might consider developing a system that parses input sentences and produces a sequence of RDF triples encoding the information being present in said input sentence. The aim of this research is to develop a system that can accomplish the aforementioned task. This research question coincides with one part of the WebNLG challenge 2020[2]. Therefore, the dataset made available through the WebNLG challenge 2020 has been used for conducting this research.

For answering said research question, a ML model has been developed which takes natural language sentences as input and outputs sets of RDF triples extracted from the input sentence. The ML model is a Deep Learning (DL) architecture, which consists of an Encoder-Decoder-Model, where the Encoder encodes given input sentences, while the Decoder takes the Encoder's outputs and transforms them into RDF triples. The Encoder is implemented as a *Bidirectional Encoder Representations from Transformers* (BERT) [4] model, which takes a tokenized input sentence, transforms its tokens into word embeddings, and eventually outputs one generated annotation vector per input token. An annotation vector is given by the last hidden state observed for any given input token. To account for the fact that potentially not all annotation vectors are of equal importance for predicting the next output token, Soft Attention [5] is applied on the annotation vectors each time before the Decoder predicts the next output token. The Decoder employed in this research consists of a vanilla Recurrent Neural Network (RNN) using Gated Recurrent Units (GRUs) [6], followed by a single fully-connected Neural Network (NN) layer. Any three consecutive output tokens (in non-overlapping fashion), as predicted by the Decoder, form a RDF triple. Furthermore, Teacher Forcing [5] is used to stabilize the training procedure.

For the evaluation of the developed system, the *F1* score [7] is evaluated on predicted triples. Also the metrics precision and recall are compared. The corresponding scores are compared in the three conditions, where 1 through 2, 1 through 4, and 1 through 6 triples, respectively, have to be predicted per input sentence.

In the next section, section 2, related work will be discussed. Section 3 will present both the employed dataset and the model architecture, as well as outline the evaluation procedure. Results will be presented in section 4 and discussed in section 5. Section 6 concludes this paper.

2 Related Work

There is a wide variety of research papers on summarizing text as RDF triples and information extraction from unstructured data. We have briefly reviewed the most similar and widely used approaches in this section.

[1] https://github.com/Bick95/text2rdf
[2] https://webnlg-challenge.loria.fr/challenge_2020/

2.1 Information Extraction from Report

Information extraction is the process of filling the fields and records of a database from unstructured or loosely formatted texts [2]. In a supervised way, ML methods are employed to automatically tune their own rules or parameters to maximize performance on extracting information from hand-labeled training examples. One class of ML models that has been applied to the task of information extraction is the class of Hidden Markov Models (HMMs), being finite state machines with probabilities on their state transitions and probabilities on the per state word emissions [2]. HMMs have been used for a long time. In this approach, the state of the model is assigned to different database fields, and the highest probability state path associated with a sequence of words indicates which subsequence of the words belongs to those database fields.

The aim of the authors of [1] was to create a system capable of general purpose, complete information extraction from radiology reports. They have developed an information extraction schema capable of representing the majority of the information contained in radiological reports and also demonstrated the feasibility of using a Neural Network (NN) architecture to extract said information. Their NN model was capable of extracting information from reports written in a wide variety of styles and formats. Their research paper also talked about various strengths of their approach to information extraction.

2.2 Bidirectional Encoder Representations from Transformers

Another relevant research paper is "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" [4]. BERT stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models, BERT is designed to pre-train deep bidirectional representations from unlabeled text. The researchers argued that current techniques restrict the power of the pre-trained representations, especially for the fine-tuning approaches. The recent improvements due to transfer learning with language models have demonstrated that rich, unsupervised pre-training is an integral part of many language understanding systems. The researchers generalized the above mentioned idea to deep bidirectional architectures, allowing the same pre-trained model to successfully tackle a broad set of NLP tasks.

The researchers improve the fine tuning based approach by proposing the use of BERT. It alleviates the previously mentioned unidirectionality constraint by using a "masked language model" (MLM) . The masked language model randomly masks some of the tokens from the input, and the objective is to predict the masked token's id based only on the masked token's context. Unlike left-to-right language model pre-training, the MLM objective enables the representation to fuse the left and right context, which allows for pre-training a deep bidirectional Transformer model [4]. In this context, the term *context* intuitively refers to the words surrounding the masked token.

2.3 Recurrent Neural Network for Learning Ontologies

Another relevant research paper deals with the use of Recurrent Neural Networks (RNNs) for learning ontologies [8]. The paper dwells that RNNs can be exploited in the ontology learning task, as they can handle the typical syntactic structures of encyclopedic texts, relieving the burden of having to handcraft large catalogs of rules and patterns through which sentences could be summarized in an ontology-compatible format. The researchers designed a Recurrent Neural Network based system endowed with some short-term memory capabilities through Gated Recursive Units. The advantages of using RNNs lie in their capability to capture meaningful features contained in text and the avoidance of feature engineering. Another advantage of using RNNs is their capability to be trained on large amounts of unlabeled training data.

3 Methods

First, the dataset will be introduced, followed by the explanations of processing pipeline, the employed model architecture, and the evaluation procedure.

3.1 Dataset

There were two datasets provided in the context of the WebNLG challenge 2020, one containing training data in the English language and a separate one for the Russian language. For the scope of this project, it has been decided to only use the English language dataset. This dataset includes training data items for 16 distinct DBpedia categories.

Each item in the dataset consists of a set of RDF triples to which one or more natural language text sentences correspond that verbalize the content of the corresponding triples. Elements contained in RDF triples are of type and order *subject-predicate-object*, where the subject is a uniform resource identifier, the predicate is a relation between subject and object, and the object can be an uniform resource identifier (as the subject) or a string, number or a date.

The dataset's training data partition was split into two sets: a set consisting of 85% of the data is used for training and 15% is used for validation throughout training. The second part of the overall dataset, technically designed to be used as a validation set, was instead use as this research's testing set due to the test set not being publicly available. In table 1, we summarize the number of RDF triple-sentence pairs provided by the dataset per use-case.

RDF triples per sentence	Training	Validation	Testing
1	6912	479	961
2	6261	370	875
3	6868	410	952
4	6371	382	901
5	4573	281	647
6	480	30	70
7	454	26	58

Table 1: Number of training, validation, and testing items per number of target RDF triples per input sentence. For this research, only up to 6 triples per input sentence have been used.

3.2 Processing Pipeline

For this research, no advanced pre-processing pipeline has been developed. The aim of this research was exclusively to test whether setting up a system that could generate RDF triples across a wide range of literary domains from input sentences would work at all, not aiming for state-of the art performance yet, but for gaining exploratory insights instead. For this purpose, a very simple, yet very robust model architecture had to be developed, which would have to be able to later deal with extremely diverse real-world input data, possibly containing both noise and inputs of excessive length. Therefore, no filtering was applied to the inputs presented to the system and pre-processing of inputs presented to the proposed system was limited to a bare minimum.

The only pre-processing step that has been applied to input sentences is tokenization. Since the model architecture's Encoder consists of the BERT language model, applying tokenization was done using the tokenizer originally used to train the BERT model. The maximal number of input tokens that a trained model can digest is always set to the maximal number of tokens detected in the training data presented to a model under a certain training condition, while all further trailing tokens are neglected by the model for technical reasons. Training conditions will be elaborated on in section 3.4.1.

3.3 Model Architecture

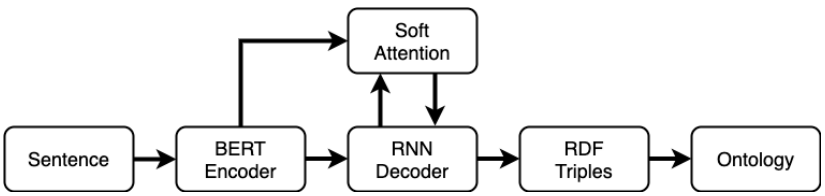

Figure 1: Schema of the model architecture

The proposed model architecture consists of three main parts and its schema is depicted in figure 1. The fist part is the Encoder, which is implemented as a pre-trained *Bidirectional Encoder Representations from Transformers* (BERT) [4] language model. The second part is a Soft Attention [5] module, which dictates what parts of the Encoder's output the Decoder shall focus on while generating the next output token. As indicated before, the third part is a Decoder, which is implemented as a simple Recurrent Neural Network (RNN) using Gated Recurrent Units [6] (GRUs) followed by a single fully-connected layer and a final Softmax [9] layer.

In more detail, this can be described as follows.

The Encoder, again being the BERT model, takes as input a tokenized natural-language input sentence. First of all, the input tokens get converted into word embeddings shipped with the utilized BERT model. The corresponding sequence of word embeddings is then sequentially fed through the BERT model and the final hidden state observed for every input token is output by the model as a so-called annotation vector. Thus, the Encoder's output encompasses one annotation vector per input token. Both the BERT model and the tokenizer are provided by [10].

The Soft Attention module's task is to take the set of annotation vectors produced by the Encoder for a given input sentence, as well as the Decoder's previous hidden state, and to output attention weights, with which the Decoder can then compute a so-called context vector, i.e. a weighted average over all annotation vectors for a given input sentence.

The corresponding weights, to produce the annotation vectors' weighted average, are determined in the Soft Attention module as a function of the annotation vectors themselves and the Decoder's previous hidden state. Concretely, a fully-connected Neural Network (NN) layer is used to produce the relative scores associated with each annotation vector and a Softmax layer turns the relative scores into normalized weights summing to 1. The goal of using this Soft Attention module is to reflect the circumstance that potentially not all annotation vectors, being related to distinct parts of the original input sentence, are of equal importance for producing the next output token to be predicted by the Decoder. The Soft Attention module thus predicts the relative importance of each annotation vector extracted from an input sentence given the information about the sequence of output tokens predicted so far, which is encoded implicitly in the Decoder's previous hidden state given as input to the Soft Attention module.

Given its previous hidden state, the previously predicted word token's word embedding, and the freshly computed attention weights produced by the Soft Attention module, the Decoder computes a probability distribution over all word tokens in the output vocabulary. The token with the highest probability is chosen as a predicted token, where any three consecutive predicted output tokens (in non-overlapping fashion) form a RDF triple. For producing the probability distribution over the output vocabulary, the Decoder first produces the context vector by computing the weighted average of the annotation vectors (given the attention weights produced by the Soft Attention module) and then feeds resulting context vector, concatenated by the word embedding of the previously generated output token, through the GRU layer for one time step. The GRU layer's output is then fed through a single fully-connected layer to get a relative score per output token. Then, the Softmax operation is applied to the previously computed scores, resulting in the aforementioned Decoder's output probabilities.

Models are trained end-to-end using back-propagation [11] and mini-batch training with a mini-batch size of 32. The Adam optimizer is used, as well as the negative log-likelihood loss function [12]. The Encoder is trained with learning rate 0.00001, while both the Decoder and the Soft Attention module are trained using a learning rate of 0.0001. A higher learning rate for the two latter modules is chosen since these modules have to be trained from scratch, while a lower learning rate is chosen to train the pre-trained Encoder in order not to risk diminishing its literary generality by potentially overfitting it to our training data which could otherwise happen when applying too much fine-tuning. To stabilize the training procedure, with decaying probability, Teacher Forcing [5] is applied to the Decoder. The initial probability of applying Teacher Forcing is set to 1 and is then, every epoch, linearly decreased by 0.05 until it reaches its minimal value of 0 probability. Teacher Forcing refers to the strategy of injecting the reference-solution's ground-truth output token, which had to be predicted in the previous time step, into the Decoder as the output token it allegedly predicted in the previous time step. Moreover, as suggested in the literature [5], the Decoder's hidden state is always initialized to the mean over all annotation vectors for a given input sentence. The word embeddings passed to the Decoder initially is always set to a dedicated *START*-token.

Layer-wise, the model architecture looks as follows. Again, a pre-trained implementation of the BERT model is used [10] as the Encoder. The Soft Attention module consists of only one fully-connected layer, which takes inputs of dimension $N \times (A + H)$ and outputs vectors of dimension $N \times 1$, where $A = 768$ refers to the number of elements per annotation vector and $H = 768$ refers to the number of nodes in the hidden layer of the Decoder's RNN. Note that H had to coincide with A, since the Decoder's hidden state of size H always gets initialized to the mean computed over multiple annotation vectors of length A each. N is variable across training conditions and is always set to the maximal number of annotation vectors that the system can digest given an input sentence. And this number, in turn, as indicated above, is equivalent to the maximal number of tokens found in any of the input sentences in the training data.

The Decoder's input layer takes input vectors of length $I = A+E$, where $E = 300$ refers to the dimensionality of the output tokens' word embeddings. The dimensionality of the RNN's output layer is also 768, coinciding with the RNN's hidden state's dimensionality. Furthermore, 20 percent dropout is used inside the RNN. The RNN's output is then fed through a Rectified Linear Unit [13] (ReLU) nonlinearity layer, before it is passed to a consecutive fully-connected layer of dimension $H \times O$, where O is variable and depends on the number of distinct output tokens being present in the training-data's target triples (plus a fixed number of additional tokens, encompassing the output vocabulary's *START*- and *END*-tokens). Here, tokens are defined as the three constituting elements contained in each target triple.

The entire system has been implemented in PyTorch.

3.4 Evaluation Procedure

Evaluation is performed both quantitatively and qualitatively.

3.4.1 Quantitative

For the quantitative evaluation, a number of test conditions is compared with respect to the *F1* score computed on the partition of the dataset reserved for testing. Also precision and recall scores are reported. The *F1* score computed for a given model on the testing data then indicates the expected quality of a respective trained model on novel data.

Formula 1 shows how the *F1* score is computed.

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{1}$$

Precision and recall, in turn, are computed as follows.

$$precision = \frac{TP}{TP + FP} \tag{2}$$

$$recall = \frac{TP}{TP + FN} \tag{3}$$

In formulas 2 and 3, TP stands for *True Positive*, FP stands for *False Positive*, and FN stands for *False Negative*. These three metrics are, in the scope of this research, defined as follows.

If a triple has been predicted for a single input sentence and the predicted triple is part of the reference solution for the given input sentence, then this triple qualifies as a *True Positive*. If a triple has been predicted and is not part of the reference solution for a given input sentence, then the predicted triple qualifies as a *False Positive*. Furthermore, also any duplicate prediction of a triple being present in the reference solution does qualify as *False Positive*. If a triple was supposed to be predicted, but the trained system missed to predict said triple given some input sentence, then the triple being absent from the prediction qualifies as a *False Negative*. Under all training conditions, systems are trained on predicting variable numbers of triples per input sentence. Since getting predictions right is of equal importance across all numbers of triples to be predicted per input sentence, no distinction is made during evaluation for how many triples per sentence TPs, FPs, and FNs have been encountered in a trained system.

Furthermore, for technical reasons, a system has to learn to predict *END*-tokens whenever it intends to indicate that it has finished producing a set of triples corresponding to a given input sentence. For the evaluation, however, the count of correctly predicted *END*-tokens does not increase a system's number of TPs, but, instead, the number of correctly predicted *END*-tokens is excluded from the number of TPs reported in this research. This is done for two reasons. Firstly, adding the number of correctly predicted *END*-tokens would artificially increase the number of correctly predicted triples per input sentence, which might lead to a too optimistic estimate of the true number of TPs that one could expect to obtain form a trained system in the future. Secondly, in spite of not explicitly reporting the number of correctly predicted *END*-tokens, a system's ability to correctly predict these tokens is implicitly assessed in the number of FPs and FNs anyway. To see why this is the case, consider the following. If a system would notoriously omit predicting an *END*-token, this would inevitably lead to many unnecessary triples being predicted. Therefore, the number of FPs would increase, leading to diminished performance scores of the considered system. On the other hand, since the generation of triples given some input sentence stops as soon as an *END*-token is predicted, if a system predicts *END*-tokens too early for given input sentences, that would result in an increased number of FNs, since the system would miss to predict a lot of required triples. Also this would result in diminished performance measures.

Throughout the quantitative evaluation, all the reported measures per training condition are averaged over three training runs per condition. Conditions, again, refer to range of how many triples per input sentence a system has to be able to predict. In the first condition, a system is trained on predicting 1 through 2 triples per input sentence, while systems are trained on predicting 1 through 4 triples per input sentence in a second condition. In a third condition, a system is trained on predicting 1 through 6 triples per input sentence. Under all conditions, systems receive 5000 mini-batch weight updates per number of triples to be predicted per input sentence for a given system. For convenience, we say that, in one epoch of training, a system receives one mini-batch weight update for every distinct number of triples it has to be able to predict (per input sentence). The number of triples a system has to be able to predict depends on the system's training condition.

Also, the quantitative analysis will plot the development of precision, recall, and *F1* scores that the trained systems have obtained on validation data (which was not part of the training data) throughout training. Sudden break-downs of these curves could potentially indicate overfitting-situations.

3.4.2 Qualitative

For the qualitative analysis, a few subjectively representative examples of triples produced by the system will be compared to reference solutions. This is to indicate potential semantic short-comings and strengths of the outputs produced by the proposed model architecture.

4 Results

In the following section, the results are going to be presented. First, quantitative results will be treated, followed by the qualitative ones.

4.1 Quantitative

Figure 2 shows both the evolution of precision scores obtained on validation data throughout training and the precision scores obtained by models trained for 5000 epochs under the three compared training conditions (see section 3.4 or figure 2 for a rehearsal of the training conditions). As can be seen, all three curves showing the evolution of precision scores per train condition start increasing after some initial stagnation. Throughout the entire training process, the validation scores obtained for the condition with 1 through 6 target triples per input sentence achieve the highest scores. The condition with 1 through 4 and 1 through 2 triples per sentence rank second and third, respectively. As can also be observed, precision scores obtained in test data are generally lower than those obtained on validation data. In spite of that, the relative ranking of conditions is the same in both the validation and the testing setting. The highest testing precision, being 0.4958, is obtained in the condition with 1 through 6 triples per sentence. It needs to be mentioned that all aforementioned scores, as well as those reported below, are obtained by averaging over three independent training runs per training condition. The associated standard deviations per training condition are indicated in pale shades of a training run's associated color. The fact that there are no pale colors visible for the test conditions with 1 through 4 and 1 through 6 triples per sentence indicates comparatively low standard deviation scores for these conditions. Furthermore, towards the end of training, after approximately 4000 epochs, the blue line associated with 1 through 2 triples per sentence exhibits some saturation behavior.

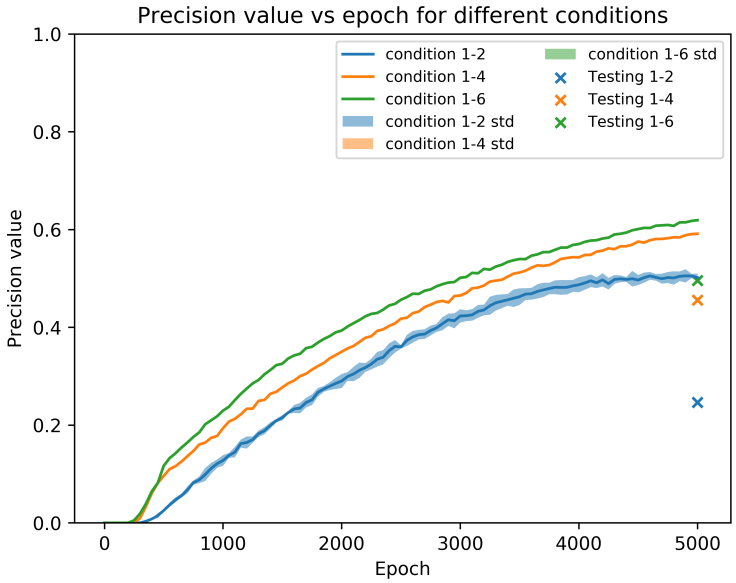

Figure 2: The intense blue, orange, and green lines show the evolution of precision scores (y-axis) obtained on validation data throughout training (x-axis in terms of epochs) with 1 through 2, 1 through 4, and 1 through 6 triples per input sentence, respectively. Since these findings have been averaged over three independent training runs per condition, the areas drawn in corresponding pale colors show the corresponding standard deviations per test condition. The blue, orange, and green crosses indicate the precision scores obtained after 5000 epochs of training on testing data for the three aforementioned conditions, respectively.

Figure 3 shows both the evolution of recall scores obtained on validation data throughout training and the recall scores obtained by the models trained for 5000 epochs under the three compared training conditions. The blue line, showing the evolution of recall scores for the condition where only 1 through 2 triples have to be predicted per sentence, exhibits a more linear increase than the lines associated with the other two training conditions. The blue line is also the first to exhibit plateauing behavior after approximately 4000 epochs. For the other two curves, plateauing is much less pronounced even after 5000 epochs of training. An interesting observation is the fact that the condition with 1 through 2 triples being predicted per sentence has the highest performance in terms of recall on the validation data after 5000 epochs of training, but the lowest performance in terms of recall on testing data. The other two conditions are very close to one another in terms of recall. The highest recall score on testing data is achieved when predicting 1 through 6 triples per input sentence. Here, recall is 0.6613.

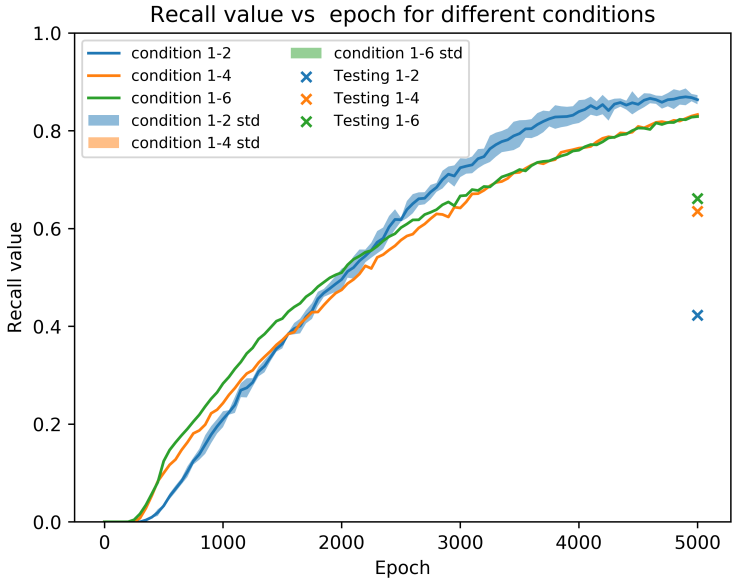

Figure 3: The intense blue, orange, and green lines show the evolution of recall scores (y-axis) obtained on validation data throughout training (x-axis in terms of epochs) with 1 through 2, 1 through 4, and 1 through 6 triples per input sentence, respectively. Since these findings have been averaged over three independent training runs per condition, the areas drawn in corresponding pale colors show the corresponding standard deviations per test condition. The blue, orange, and green crosses indicate the recall scores obtained after 5000 epochs of training on testing data for the three aforementioned conditions, respectively.

The last figure, figure 4, shows both the evolution of *F1* scores obtained on validation data throughout training and the *F1* scores obtained by models trained for 5000 epochs under the three compared training conditions. Since *F1* scores are a combination of the former two measures precision and recall (see equation 1), many findings reported for the former two figures can be observed from this figure as well. As reported for figure 3 already, the increase of the *F1* metric for the condition where 1 through 2 triples have to be predicted per sentence looks largely linear, but plateauing behavior can be observed after approximately 4000 epochs of training. The slope of the other two curves looks less linear again, but they reliably yield *F1* scores being higher than those associated with the former condition. Again, the condition where 1 through 6 triples have to be predicted per sentence yields the highest results in terms of *F1* scores on both validation and testing data, analogous to what could be observed in figure 2 in terms of precision already, are highest for the condition where 1 through 6 triples have to be predicted. The corresponding *F1* score is 0.5667. The second-highest *F1* scores are observed for the condition in which 1 through 4 triples have to be predicted per sentence.

4.2 Qualitative

Mismatch causes	Target RDF triples	Predicted RDF trip
PAD token	['A.F.C.Fylde', 'ground', 'PAD'] ['Angola, Indiana', 'isPartOf', 'PAD'] ['A.E Dimitra Efxeinoupolis', 'fullname', 'PAD'] ['Jens Härtel', 'clubs', 'PAD']	['A.F.C.Fylde', 'ground', 'Warton,Fylde'] ['Angola, Indiana', 'isPartOf', 'SteubenCounty,Indiana'] ['A.E Dimitra Efxeinoupolis', 'fullname', '1500'] ['Jens Härtel', 'club', 'Berliner AK 07']
Close match	['John van den Brom', 'managerClub', 'AZ Alkmaar'] ['Stuart Parker (footballer)', 'club', 'Bury F.C.'] ['United Kingdom', 'leader', 'Elizabeth II'] ['Apollo 14', 'operator', 'NASA'] ['Amarillo, Texas', 'isPartOf', 'United States']	['John van den Brom', 'club', 'AZ Alkmaar'] ['Stuart Parker (footballer)', 'team', 'Bury F.C.'] ['United Kingdom', 'leaderName', 'Elizabeth II'] ['Apollo 12', 'operator', 'NASA'] ['Amarillo, Texas', 'country', 'United States']
Numbers	['AZ Alkmaar', 'capacity', '17023'] ['AC Hotel Bella Sky Copenhagen', 'floorCount', '"23" xsd:...']	['AZ Alkmaar', 'capacity', '"17023" xsd:...'] ['AC Hotel Bella Sky Copenhagen', 'floorCount', '23']

Table 2: Representative examples categorized accordingly to the main reasons for the difference between target RDF triples and respective predicted RDF triples.

There are many reasons why the predicted RDF triples differ from the target ones. One of the most commonly found reasons is that if the target RDF triple contains an item that was not present during the training phase, said item is replaced with a 'PAD' token in the target triple, which the system will not able to correctly predict. PAD tokens are

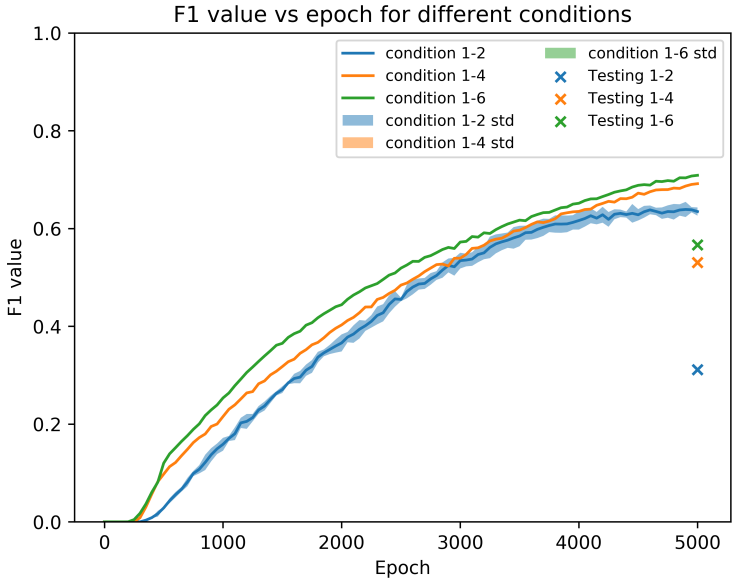

Figure 4: The intense blue, orange, and green lines show the evolution of *F1* scores (y-axis) obtained on validation data throughout training (x-axis in terms of epochs) with 1 through 2, 1 through 4, and 1 through 6 triples per input sentence, respectively. Since these findings have been averaged over three independent training runs per condition, the areas drawn in corresponding pale colors show the corresponding standard deviations per test condition. The blue, orange, and green crosses indicate the *F1* scores obtained after 5000 epochs of training on testing data for the three aforementioned conditions, respectively.

most often used to replace the *object* item in a RDF triple, and are used in particular for replacing numbers, but also proper names and other nouns. The system, in most cases, seems to infer correctly the type of content a PAD token is hiding. For example, if, in the original sentence, the PAD token was associated with a number (not being part of the training vocabulary), then the system will predict an incorrect number or, if the token was proper noun, it will predict an incorrect proper noun. However, in a very small number of cases, the item corresponding to the PAD token is predicted correctly. One example of this case, in table 2, is the sentence *'Angola is in Pleasant Township which is part of Steuben County, in Indiana.'* which has as target RDF triple ['Angola, Indiana', 'isPartOf', 'PAD'] and the predicted RDF triple is ['Angola, Indiana', 'isPartOf', 'Steuben County, Indiana']. In this case, the system is able to predict the correct object. So, compared to the original sentence, the predicted RDF triple is correct, but due to the object item being compared to the PAD token it results in a FN.

Another reason is that, in some cases, the target RDF triple and the predicted RDF triple have one item (of the same type) that is similar in meaning but it's not an exact match. This causes the predicted RDF triple to be incorrect and thus a FN. In table 2, some of these occurrences are shown. For instance, the sentence *'The leader of the United Kingdom is Elizabeth II.'* has the target RDF triple ['United Kingdom', 'leader', 'Elizabeth II'], however the predicted triple is ['United Kingdom', 'leaderName', 'Elizabeth II']. In this case, the predicate item of the triple is incorrect as it's not an exact match, however the actual meaning is the same and the predicted RDF triple is semantically correct. Similarly, for the sentence *'John van den Brom plays at the AZ Alkmaar club.'*, the target triple is ['John van den Brom', 'managerClub', 'AZ Alkmaar'], however the predicted triple is ['John van den Brom', 'club', 'AZ Alkmaar']. In this case, the predicted triple seems to be a more correct match to the original sentence as it does not state anywhere in the input sentence that the subject is a manager, but just that the given person plays at a given club.

By eyeballing the results, we also looked at the more complex sentences. It was found that, if a sentence is translated as two (or more) target RDF triples then, usually, at least one predicted RDF triple is correct, but not all of them, thus resulting in a TP for the one RDF predicted correctly and a FN for the RDF not present in the prediction. If the complex sentence is instead translated as one target RDF triple, then the items are quite long and involve many different words. In this case, the predictions seem to be correct in most cases.

Lastly, inconsistency in the provided target triples was detected. Some numeric tokens contain in addition to the number itself a trailing specification of the type of number, while other numeric tokens do not contain such a type specification. Since the presence (or or absence) of this specification is not consistent across all numbers, even if the predicted RDF item is the same number, it is not recognized as such because it is written in two different formats. Thus, due to inconsistency in the target triples, some predicted RDF triples are not recognized as being correct (see table 2).

5 Discussion

As reported in the previous section, the proposed method was able to obtain *F1* scores of 0.5667 on previously unseen data in the best examined condition, which is the condition in which 1 through 6 triples had to be predicted per input sentence. Given how minimally pre-processed input data fed into the system was and how conceptually simple the proposed model architecture appears to be, these results seem very promising. They raise the question how much better performance could potentially be when extending the proposed model architecture, thereby increasing the model's complexity, and thus its ability to incorporate even more complex sentence structures which the system might have been incapable to learn given its current model complexity limitations.

Speaking about limitations, from the plateauing behavior of the blue curves associated with the training condition where 1 through 2 triples had to be predicted per sentence, it becomes apparent that the models trained under this condition were not able to extract much more information from the given training data after 5000 epochs, since otherwise the curve would not have plateaued. Judging from the fact that models trained under the other conditions, where even more complex sentence structures had to be dealt with, did not exhibit plateauing behavior yet after 5000 epochs, it can be concluded that the proposed model's complexity was very likely not the limiting factor causing the model's inability to continue training after 5000 epochs in the condition where 1 through 2 triples had to be predicted. In other words, seemingly, there was too little diverse training data provided for the aforementioned training condition to achieve even higher scores since models in said training condition did not continue to increase performance measured by the *F1* score in spite of the model complexity being rich enough to successfully deal even with more complex training data under other conditions. However, also for the other two curves, associated with the other two training conditions, the increase in performance seemed to decrease, such that also there plateauing behavior is expected to appear soon after having trained systems under these conditions for 5000 epochs already. Thus, as usual, training the proposed system on more training data is expected to increase its performance even further in all three conditions.

The fact that the evaluation scores computed on testing data are so much lower than those computed on validation data could be due to the following reason. While it has been made sure that all combinations of sentences and their associated triples, of which at least one combination has been reserved for validation, have been removed from the actual training data, it could still be that parts of the overall training data partially coincide with the validation data across different numbers of triples per input sentence. In other words, a lengthy sentence in the evaluation data could contain a sub-string, being another training sentence of shorter length, which has actually been reserved for validation. Consequently, training and validation data could be partially shared, such that the corresponding validation scores might constitute a rather optimistic estimate of the expected performance of an evaluated classifier. It is also the case that some tokens in the testing data were not present in the training data (as discussed in the previous section) and thus the system was not able to predict them. This, together with the inconsistency found in the numeric tokens, could have also lowered the evaluation scores.

Furthermore, in the proposed processing pipeline, no masking has been applied. As discussed in the qualitative results, one problem was that the trained models sometimes predicted incorrect numeric tokens, while still being able to correctly decide that a predicted token has to be numeric in general. Here, masking of numeric tokens could potentially have alleviated the aforementioned problem. Masking could have simplified the learning task in two ways. Firstly, masking (among others) numeric tokens would have decreased the vocabulary size by removing many numeric tokens from the output vocabulary, and thus would have simplified the training procedure. Secondly, in the current setting, any numeric token receives only a small portion of training, since any of the many distinct numeric tokens appears only very few times throughout the training examples. By applying masking, a shared numeric token mask would have received a lot of training, since all numeric tokens present in the training dataset would have contributed to training its correct usage. Therefore, a model trained on such a masking token could possibly have worked much more robustly in the presence of numeric tokens. However, using masking would not have simplified the overall researchers' development task. It would merely have shifted the problem of having to learn correct mappings of numbers from text to triples from a supervised prediction task to one which might otherwise have involved using handcrafted number extraction rules to post-process outputs produced by the proposed system. Therefore, the implementation used in this very exploratory research has omitted the use of masking. A potential solution for both avoiding the aforementioned post-processing step and reducing the output vocabulary size in general could have been to split target triples to be predicted by a system into sub-word tokens instead of splitting them into word tokens. Thus, each word contained in a target triple could have been split into multiple lexical sub-components. In this way, each component constituting the target vocabulary could have been re-used for predicting a multitude of possible words constituting the triples to be predicted. As indicated above, this solution could have even simplified the problem of mapping masked entities to triples, since masks could have been decomposed into both a general-purpose mask-placeholder and a mask-ID. Then, the model could have been trained on mapping masked entities encountered in input sentences to triples by always predicting both mask-compounds, i.e. the placeholder and its ID, in a sequence to allow for a straight-forward substitution of masks by their masked contents afterwards again.

Moreover, to reduce the number of hyperparameters to be tuned by the researchers, it has been decided throughout this research to make some parameters, e.g. the maximal number of input tokens a model can digest, dependent on

statistics derived from the training dataset. Pre-determining more reasonable, stricter maximal bounds on such hyperparameters could be a potential improvement of the presented model architecture as well, preventing excessively large hyperparameters being chosen due to severe noise or outliers potentially present in the training data.

6 Conclusion

The aim of this research was to develop a system that would be able to perform the task of summarizing sentences expressed in natural language as sets of RDF triples, and hence in a structured form. Primarily, this research aimed at producing a system acting as a proof of concept that such a task could be solved even across a set of different literary domains, where many of those were covered to some extent by the employed dataset. Indeed, given the promising findings reported above, we accomplished the task of coming up with a system that could work to some extent even on sentences containing information to be expressed in up to 6 triples. Considering the use of a simple model architecture, a relatively small dataset, and comparatively low training times of up to 12 hours only for the most exhaustive training runs, these findings suggest a vast potential for improvement by using a more sophisticated model architecture and training it for more iterations on a larger dataset. As previously discussed, also further improvements with respect to the pre-processing pipeline could lead to better results and are up to future research.

References

[1] J. M. Steinkamp, C. Chambers, D. Lalevic, H. M. Zafar, and T. S. Cook, "Toward complete structured information extraction from radiology reports using machine learning," *Journal of Digital Imaging*, vol. 32, no. 4, pp. 554–564, 2019.

[2] A. McCallum, "Information extraction: Distilling structured data from unstructured text," *Queue*, vol. 3, no. 9, pp. 48–57, 2005.

[3] J. M. Vieira and A. Ciula, "Implementing an rdf/owl ontology on henry the iii fine rolls.," in *OWLED*, Citeseer, 2007.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[5] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, pp. 2048–2057, 2015.

[6] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[7] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.

[8] G. Petrucci, C. Ghidini, and M. Rospocher, "Using recurrent neural network for learning expressive ontologies," *arXiv preprint arXiv:1607.04110*, 2016.

[9] R. Zunino and P. Gastaldo, "Analog implementation of the softmax function," in *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No. 02CH37353)*, vol. 2, pp. II–II, IEEE, 2002.

[10] 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. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Huggingface's transformers: State-of-the-art natural language processing," *ArXiv*, vol. abs/1910.03771, 2019.

[11] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*, pp. 65–93, Elsevier, 1992.

[12] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[13] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.