Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020)
Main Track

# Seq2KG: An End-to-End Neural Model for Domain Agnostic Knowledge Graph (not Text Graph) Construction from Text

**Michael Stewart** and **Wei Liu**
The University of Western Australia
michael.stewart@research.uwa.edu.au, wei.liu@uwa.edu.au

**Abstract**

Knowledge Graph Construction (KGC) from text unlocks information held within unstructured text and is critical to a wide range of downstream applications. General approaches to KGC from text are heavily reliant on the existence of knowledge bases, yet most domains do not even have an external knowledge base readily available. In many situations this results in information loss as a wealth of key information is held within "non-entities". Domain-specific approaches to KGC typically adopt unsupervised pipelines, using carefully crafted linguistic and statistical patterns to extract co-occurred noun phrases as triples, essentially constructing text graphs rather than true knowledge graphs. In this research, for the first time, in the same flavour as Collobert et al.'s seminal work of "Natural language processing (almost) from scratch" in 2011, we propose a Seq2KG model attempting to achieve *"Knowledge graph construction (almost) from scratch"*. An end-to-end Sequence to Knowledge Graph (Seq2KG) neural model jointly learns to generate triples and resolves entity types as a multi-label classification task through deep learning neural networks. In addition, a novel evaluation metric that takes both semantic and structural closeness into account is developed for measuring the performance of triple extraction. We show that our end-to-end Seq2KG model performs on par with a state of the art rule-based system which outperformed other neural models and won the first prize of the first Knowledge Graph Contest in 2019. A new annotation scheme and three high-quality manually annotated datasets are available to help promote this direction of research.

## 1 Introduction

Automatic construction of knowledge graphs directly from text has attracted considerable attention from the research community in recent years (Wu et al. 2019). Knowledge graphs encapsulate valuable information, typically extracted from large collections of textual data, and hence critical in providing background semantic knowledge in a variety of tasks such as question answering and knowledge discovery (Kertkeidkachorn and Ichise 2017; Stewart et al. 2017), and more recently in vision and language tasks such as automatic image captioning (Zhou, Sun, and Honavar 2019) and video description (Vasile and Lukasiewicz 2018).

Knowledge graph construction (KGC) from text traditionally involves two primary stages. *Information extraction* yields triples that identify subject and object entities, connecting them through relations (Allahyari et al. 2017). Most systems handle this task using a pipeline of open information extraction, named entity recognition and relation extraction (Martinez-Rodriguez, Lopez-Arevalo, and Rios-Alvarado 2018). *Entity linking* then resolves the entities and relations to concepts in a knowledge base such as DBpedia (Mehta, Singhal, and Karlapalem 2019).

This traditional approach to constructing a knowledge graph from entities and relations linked to concepts in a knowledge base presents two key issues. Firstly, many domains feature entities that do not exist in a knowledge base (Chen et al. 2018), and often cannot be labelled using standard named entity classes. Secondly, a wealth of information is lost when important information is held within phrases that do not conform to a specific known entity class. Take the following as an example:

> *Retailers in Phoenix , Ariz., say P&G 's new powdered detergent – to be called Cheer with Color Guard – will be on shelves in that market by early November.*

Traditional KGC techniques would only capture the fact that *Cheer with Color Guard* is a product of *P&G*. No information is captured on the fact that it is a detergent or that it will be released in early November, as shown in Figure 1.

We therefore argue that in order to provide a general, domain agnostic solution for KGC from text, it is beneficial to approach KGC without the assumption of a knowledge base. The head and tail of each triple should be comprised of any terms carrying important information, regardless of whether they are named entities or not. The attributes of each node (i.e. the entity type) as well as the relations can be determined through the text itself, alleviating the need for a knowledge base. These attributes serve to encapsulate the knowledge held within the text, differentiating our approach as true knowledge graph construction as opposed to text or data graph extraction. Our method is generalisable to a range of domains and excels when important information is held within "non-entities" or when a domain-specific knowledge base is not available.

Existing text graph construction systems, despite the similarity with ours in their knowledge-base free nature, need complex pipelines purposely built for the target domains such as education (Chen et al. 2018), geoscience (Wang et al. 2018), and industry-specific news (Stewart, Enkhsaikhan,

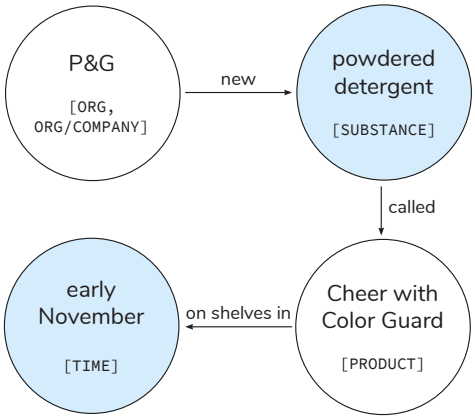


Figure 1: A knowledge graph constructed from text that encapsulates useful information without requiring an external knowledge base. Node attributes (i.e. entity type) are denoted with square brackets. Light blue indicates that a node would not appear when traditional KGC from text is applied.

and Liu 2019). They are hence not domain agnostic. Furthermore, text graphs do not store any node attributes, meaning they cannot be considered true knowledge graphs. Predicting triples and node attributes from text directly using deep learning would offer a solution to these issues, but there is not yet any suitable data format for training an end-to-end neural model to directly predict a set of triples and their associated attributes from text.

Another key issue in existing KGC from text research is a lack of evaluation metrics for automatically evaluating the quality of triples produced by a KGC system. State-of-the-art KGC research employs qualitative manual evaluation (Mehta, Singhal, and Karlapalem 2019) as the development of graph evaluation metrics is still an ongoing area of research.

In light of these issues we first introduce *a data annotation scheme* that enables the training of end-to-end deep learning models for knowledge graph construction from unstructured text. The significance of this work is in the same flavour as "Natural language processing (almost) from scratch" (Collobert et al. 2011). So long as the training data is prepared according to our proposed annotation scheme, any sequence labelling model can be used to construct knowledge graphs directly from text, almost from scratch.

Then we present two neural models: a filtering model, and an end-to-end model. The end-to-end model takes text as a sequential input and jointly performs triple extraction and entity typing to determine node attributes, which is capable of *turning a sequence into a knowledge graph* (Seq2KG). The triples are considerably more fine-grained and applicable to a wide variety of domains when compared to named entity recognition.

Existing graph evaluation metrics typically target either structure or semantic similarity, but hardly both. For example, You et al. (2018) use maximum mean discrepancy to measure structural similarity, while Hawes and Kelleher (2004) combine relation-based and feature-based alignment to measure for measuring semantic similarity. For fairer, stricter and simple performance evaluation, we introduce *a new evaluation measure Embedding Similarity* (ESim) based on random walks and averaging word embeddings for KG comparison. ESim captures both graph-level structural information as well as word-level semantic information.

Finally, we release *three high quality annotated KGC datasets* to the research community, one of which is also labelled with entity types.

## 2 Related Work

Knowledge graph construction from text has been approached from a few different perspectives. The majority of KGC systems construct true knowledge graphs, but some more domain-specific KGC systems instead construct text graphs. The primary difference between the two is that the knowledge graph captures some form of knowledge related to its nodes or relations (Wu et al. 2019), whereas text graphs do not. This knowledge capturing is performed by linking each concept in the graph to an entity or relation in an existing knowledge base.

The majority of KGC systems are built to handle clean, public domain corpora such as news reports and wiki articles. The system proposed by (Mehta, Singhal, and Karlapalem 2019) is a pipeline comprising named entity recognition, coreference resolution, triple extraction, and predicate mapping. Named entity recognition is used as a way to map entity mentions in text to their corresponding concepts in DBPedia. Triple extraction is performed using OLLIE (Schmitz et al. 2012), an Open Information Extraction (OpenIE) method. A novel method is presented for predicate mapping, whereby the relation phrases are mapped to DBPedia concepts using a hierarchical multi-stage siamese recurrent network. The vast number of triples produced by OpenIE requires the candidate triples to be filtered with the aid of the knowledge base. Consequently, the coverage of the entities is limited by the number of concepts in the knowledge base. Potential new concepts cannot be added.

Other pipeline-based systems feature vastly different organisations of components. The pipeline introduced by (Martinez-Rodriguez, Lopez-Arevalo, and Rios-Alvarado 2018) comprises sentence segmentation, part of speech tagging, syntax tree parsing, entity recognition and linking, relation extraction, and triple filtering. The systems employs OpenIE as a means to extract relations, rather than entire triples, ensuring that the heads and tails of each triple are entities present in a knowledge base. NOUS (Choudhury et al. 2017) uses a distant supervision approach to filter the triples produced by OpenIE prior to mapping the predicates to entries in a knowledge base, creating a new entry if the node or relation does not yet exist. T2KG (Kertkeidkachorn and Ichise 2017), on the other hand, comprises a pipeline of entity mapping, coreference resolution, triple extraction, triple integration, and predicate mapping. The entity mapping stage identifies named entities in the text and maps them to existing concepts in a knowledge base, creating new concepts if they are not present. Triples are constructed via

OpenIE, and the predicate mapping stage maps the relations of each triple to existing predicates in a knowledge base.

In less ubiquitous domains it is rare for KGC systems to link nodes and edges to concepts in an existing knowledge base because no such knowledge base exists. These systems hence construct text graphs, linking one entity or noun phrase to another via a relation. (Wang et al. 2018) construct text graphs from Chinese geoscience literature. The nodes and edges in the graph are comprised of noun phrases that are segmented using a conditional random field (CRF). (Chen et al. 2018) target the education domain, extracting education-related concepts (as opposed to entities) as nodes using a CRF. Relations are extracted using association rule mining. (Stewart, Enkhsaikhan, and Liu 2019) construct text graphs from industry-specific news using a complex pipeline comprising many NLP techniques such as tokenisation, part-of-speech (POS) tagging, named entity recognition, coreference resolution, and noun/verb phrase chunking.

In summary, it is common for KGC systems to link entities and relations to concepts in a knowledge base when constructing a KG from a domain whereby such a knowledge base is readily available. This is not possible in domain-specific applications, however. Every system is built from a pipeline designed to handle a specific domain, but there does not yet exist a KGC system capable of encapsulating knowledge from a wide variety of domains.

# 3 Models for Domain-independent Knowledge Graph Construction from Text

We hereby introduce three models[1] for knowledge graph construction: a rule-based method and two neural models. The goal of each method is to produce a set of triples that represent a document in the form of a knowledge graph. For example, if given the following document:

> *We've known for some time now that Lamborghini is making an SUV. Called the Urus , it is rumored to be powered by a twin-turbo 4.0-liter V8 generating 600 hp.*

Our models aim to produce the triples:

- *(Lamborghini, making, SUV)*
- *(SUV, Called, Urus)*
- *(Urus, powered by, twin-turbo 4.0-liter V8)*
- *(twin-turbo 4.0-liter V8, generating, 600 hp)*

In addition, our End-to-End Seq2KG model also labels the entity types of the head and tails of each triple, assigning `[ORGANISATION, ORGANIZATION/COMPANY]` to *Lamboghini*, `[PRODUCT, PRODUCT/CAR]` to *Urus*, and so on. In this way it is able to capture knowledge directly from text without requiring an external knowledge base.

The embedding layers of the deep learning-based models use BERT (Devlin et al. 2018) to encode the input sequences. BERT achieves strong performance on a wide variety of NLP tasks and is able to encapsulate contextual information more effectively than many other embedding techniques such as Word2Vec (Mikolov et al. 2013) as it generates embeddings for each token with respect to the context in which it appears.

[1]The source code of each model is available on GitHub: https://github.com/Michael-Stewart-Webdev/Seq2KG

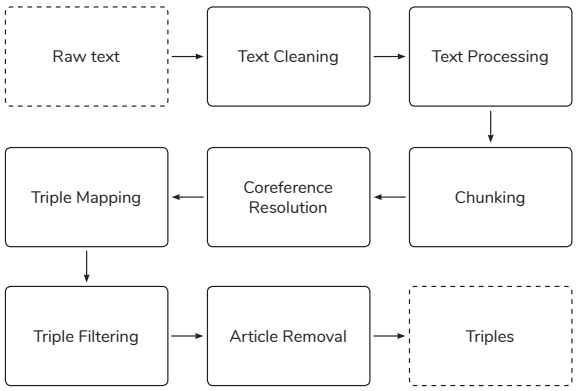


Figure 2: A diagram of the core components of our rule-based method for triple extraction.

## 3.1 Rule-based Model

The rule-based model, first introduced as a method for extracting triples from automotive and catering news datasets (Stewart, Enkhsaikhan, and Liu 2019), uses a pipeline-based approach in order to convert a document into a set of triples. It comprises seven distinct stages, as shown in Figure 2, each of which may be adjusted to suit a particular domain. This rule-based pipeline outperformed other neural models at the time and won the first prize of 2019 ICDM/ICBK Knowledge Graph Contest[2]. Therefore, we use it as a pseudo-upperbound to match other models' performance against. In other words, a neural model without domain restriction achieving similar or better performance than this rule-based model will be considered as the new state of the art.

**Text cleaning**: Text data is cleaned to manage special characters such as hyphen and quotation marks and also break sentences joined together with no space between them.

**Text processing**: The text is processed through tokenisation, POS tagging, entity recognition and dependency parsing steps using SpaCy[3].

**Chunking**: Noun phrases (NPs) and verb phrases (VPs) are chunked. Noun chunks are phrases that have a noun and the words describing the noun. For example, *an American multinational automaker* and *a suburb of Detroit*. Action words are also chunked so that verb phrases can contain verbs, particles and/or adverbs that represent more meaningful relations between entities. For example, *was founded by* and *incorporated on*.

**Coreference Resolution**: A list of coreferenced items is created using NeuralCoref[4]. For our example the one coreference item is identified, i.e. the term *it* referring to *Urus*.

[2]http://icdm2019contest.mlamp.cn/

[3]https://spacy.io/

[4]https://github.com/huggingface/neuralcoref

**Algorithm 1** Chunking of noun phrases and verb phrases

**procedure** CHUNKPHRASES(*document*)
  **for each** *sentence in document* **do**
    ▷ Chunk noun phrases (NPs) and tag as *ENTITY*
    **chunk** *NPs* ▷ NP
    **chunk** $'('+NP+')'$ ▷ (NP)
    **chunk** $NP+' of'+NP$ ▷ NP of NP
    **chunk** $NP+NP$ ▷ NP NP
    ▷ Chunk verb phrases and tag as *VERB*
    **chunk** $VERB+PART$ ▷ verb + particle
    **chunk** $VERB+ADP$ ▷ verb + adpositions
    **chunk** $ADP+VERB$ ▷ adpositions + verb
    **chunk** $PART+VERB$ ▷ particle + verb
    **chunk** $VERB+VERB$ ▷ verb + verb
  **return** *document* ▷ Document with phrase chunks

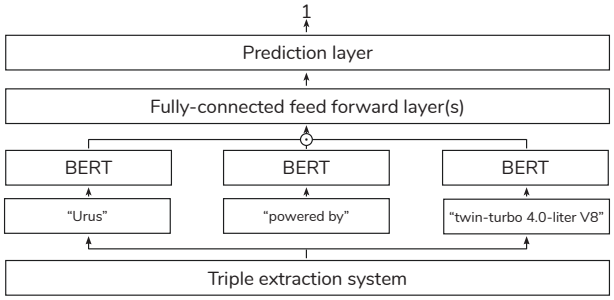


Figure 3: The filtering model. $\odot$ denotes concatenation.

Coreference items are resolved on the triples by replacing the original phrase with the referred phrase for each item. For example, *it* will be replaced by *Urus*.

**Triple Mapping**: Triples are created from the sentences in *head, relation, tail* format using Algorithm 2. First, head and tail entities are extracted with their relations from the sentences and creates a list of triples. Second, a graph is created from those triples to uncover the relations among named entities in separate sentences. Based on the relations of prepositions such as *in, on, at*, more triples are created to provide more links between named entities in the graph. Finally, the triples created by these two steps are joined to make the full list of triples for the given text.

**Triple Filtering**: To improve the quality of the triples, the filtering is performed to remove any triple with a stop word as a head entity. The stop words include NLTK stop words, names of days (Monday to Sunday) and names of months (January to December).

**Article Removal**: To clean the entities we removed some tokens including articles (e.g., a, an, the), possessive pronouns (e.g., its, their) and demonstrative pronouns (e.g., that, these) from the head and tails of each triple.

## 3.2 Filtering Model

The filtering model, as shown in Figure 3, is a deep-learning based model that aims to take the output of an existing triple extraction system and identify the set of valid triples contained within. It accomplishes this by classifying each triple into one of two classes (valid and invalid) using a fully connected feed forward neural network, which can be replaced by any binary classifier. For example, if given the triples *(Urus, revealed, twin-turbo 4.0-liter V8), (Urus, powered by, twin-turbo 4.0-liter V8 )*, the model should classify the first triple as invalid (i.e. class 0), and the second as valid (class 1). All of the triples labelled as valid by the model will form the set of output triples.

**Input triples** During both training and evaluation, the input triples to the model are generated using a greedy triple extraction algorithm, which is a pipeline comprising part of speech tagging and regular expression-based noun/relation phrase chunking. These tasks are performed using NLTK[5]. Triples are constructed from each sentence by enumerating over all combinations of *(noun phrase 1, relation phrase, noun phrase 2)* up to a certain distance threshold. This method provides the model with a large sampling of triples with which to filter and identify the valid triples.

When training the model, we construct a set of ground truth triples (with a label of 1) using the rule-based method, discussed in Section 3.1. The rule-based model was found to produce better results than OpenIE because OpenIE tends to produce a vast number of irrelevant triples consisting of long phrases.

**Hidden and output layers** The first layer of the filtering model encodes the head, relation and tail of a triple using BERT. We take the embedding of each sequence via average pooling, i.e. the final vectors obtained are the average embedding across all wordpieces in each input sequence. We denote the embedding vectors for the head, relation and tail as $e_h$, $e_r$ and $e_t$ respectively. These three vectors are concatenated to form the combined representation $c$. Let $\odot : \mathbb{R}^n \odot \mathbb{R}^n = \mathbb{R}^{2n}$ denote the concatenation of two vectors:

$$c = e_h \odot e_r \odot e_t \tag{1}$$

The combined representation $c \in \mathbb{R}^{3n}$ is then fed through one or more fully connected feed forward layers, each of which use a tanh activation function. Given $A$ and $B$ are weight matrices that are learned, and $b$ is the bias:

$$x = \tanh c = cA^\top + b \tag{2}$$

The outputs of the final linear layer are then fed through one final layer, which performs a linear transformation followed by a sigmoid activation function to translate the weights back to a single value between 0 and 1. Given $B$ is a weight matrix that is learned:

$$y' = \frac{1}{1 + e^{-(xB^\top + b)}} \tag{3}$$

During the evaluation stage, the predicted value $y'$ is used to determine whether the given input is a valid triple or not. We apply a simple threshold: if $y' > t$, where $t$ is the threshold value, then the triple is considered valid and is appended to the output set for the current document.

[5]https://www.nltk.org/

**Algorithm 2** Triple mapping algorithm

**procedure** GETTRIPLES($document$)
  **for each** $sentence\ in\ document$ **do**
    $relations \leftarrow verbs + prepositions + postpositions$ ▷ Select relations such as *showcased, has, in, to, during*
    **for each** $r\ in\ relations$ **do**
      $heads \leftarrow$ *entities on the left side of* $r$ ▷ Get the head entities for the relation *r*
      $tails \leftarrow$ *entities on the right side of* $r$ ▷ Get the tail entities for the relation *r*
      **for each** $h\ in\ heads$ **do**
        **for each** $t\ in\ tails$ **do**
          $triples \leftarrow triples + [h, r, t]$ ▷ Add *[head, relation, tail]* to the list of triples
  **return** $triples$ ▷ Return the list of triples

**procedure** EXTRACTTRIPLES($document$)
  ▷ Extract triples from the document at the sentence level
  $triples \leftarrow$ GETTRIPLES($document$)
  ▷ Extract the triples at the document level using the graph shortest paths
  $G \leftarrow$ *create graph(triples)* ▷ Build a graph from the triples using NetworkX package
  $paths \leftarrow$ *get shortest paths(G)* ▷ Get all shortest paths between named entities
  **for each** $h, t\ in$ *pairs of named entities* **do**
    **if** *h and t connected by a path using 'in', 'at', 'on' prepositions* **then**
      $triples \leftarrow triples + [h,' in', t]$ ▷ Add *[head, 'in', tail]* to the list of triples
  **return** $triples$ ▷ Return the full list of triples

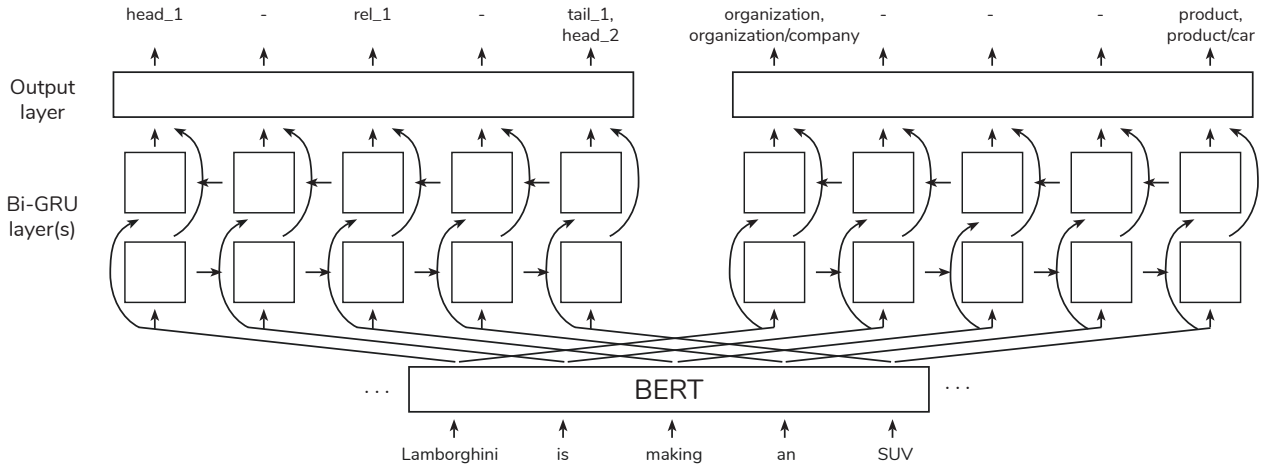


Figure 4: A diagram of our End-to-End Seq2KG model. The tokens to the left and right of the example sequence are not shown for brevity.

**Loss function** The loss of the filtering model is calculated using binary cross entropy:

$$loss = -(y' log(y) + (1 - y) log(1 - y')) \quad (4)$$

where $y$ is the correct class for the current triple (i.e. 0 or 1) and $y'$ is the value predicted by the model.

### 3.3 End-to-End Seq2KG Model

The End-to-End Seq2KG model, as shown in Figure 4, aims to extract triples directly from the text and label each node with its corresponding entity type(s) via a neural network architecture. In contrast to the rule-based method and filtering method, the End-to-End Seq2KG model requires absolutely no feature engineering. It does not need to be tailored to a specific domain and is thereby domain agnostic. While it requires annotated training data, obtaining annotated data is preferable to carefully crafting rules for a potentially "brittle" rule-based system. Moreover, the process of annotating data is a task that can be performed by anybody with knowledge of the domain, in contrast to the construction of rule-based systems which demand programming knowledge and expertise in natural language processing.

The End-to-End Seq2KG model treats the task of KGC as a multi-label sequence labelling problem: given a sentence comprised of tokens, its goal is to label each token with its membership and role in zero or more triples, as well as its entity type(s). The set of triples for the sentence is then produced based on the triple membership labels assigned to each token by the model. This allows the model to generate a dynamic number of triples per sentence.

The model utilses the Bi-directional Gated Recurrent Unit (Bi-GRU) (Cho et al. 2014), a recurrent architecture that excels at handling sequential data. The GRU converges quicker and requires fewer parameter updates than the LSTM (Chung et al. 2014). The Bi-GRU, as opposed to a unidirectional GRU, allows for both forward and backwards contexts to be taken into account when predicting the validity of a triple or the membership of a word in a triple. Note that the Bi-GRU can be replaced with any sequence labelling model, such as the transformer (Vaswani et al. 2017).

When extracting triples from documents comprised of multiple sentences, the End-to-End Seq2KG model is supplemented by a simple pre-processing step that performs *coreference resolution* on each document via Neuralcoref[6]. In our example, "it" is replaced with "Urus" prior to the training of our models.

**Hidden and output layers** The input sequence is first embedded via BERT before being sent to both the triple prediction component and entity typing component. The embeddings pass through one or more Bi-GRU layers, each of which encodes the input sequence from left-to-right and right-to-left.

The forward and backwards Bi-GRU layers serve to encode the embedded inputs and provide the final output layer with a representation that may be mapped to a sequence of label probabilities. The update gate $z$ and reset gate of the GRU $r$ are calculated as follows, where $U$ and $W$ are weight matrices to be learned, $x_t$ is the input at time $t$, and $h_t$ is the hidden state at time $t$ (Koehn 2009):

$$z_t = \sigma(x_t U^z + h_{t-1} W^z + b_z) \tag{5a}$$
$$r_t = \sigma(x_t U^r + h_{t-1} W^r + b_r) \tag{5b}$$

The hidden state of the GRU is then calculated as follows:

$$\widetilde{h}_t = tanh(x_t U^h + (r_t \odot h_{t-1}) W^h) \tag{6a}$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \widetilde{h}_t \tag{6b}$$

The outputs of this layer are then fed through one final layer, which performs a linear transformation followed by the sigmoid activation function as per Equation 3. The output vector, $y'$, contains one weight corresponding to each label $\in N$, where $N$ is the set of all labels.

**Loss function** The loss of each component is calculated using categorical cross entropy:

$$l_n = -(y_n log(y'_n) + (1 - y_n) log(1 - y'_n)) \tag{7a}$$
$$loss = \frac{\sum_{n \in N} l_n}{|N|} \tag{7b}$$

Here, $y_n \in \{0, 1\}$ is the ground truth label of the class of index $n$, $\{y'_n \in \mathbb{R} \,|\, 0 \leq y'_n \leq 1\}$ is the prediction score associated with the class of index $n$, and $N$ is the set of labels. We then average the loss of each component. Given $loss_t$ is the loss of the triple extraction component, and $loss_e$ is the loss of the entity typing component:

$$joint\_loss = \frac{loss_t + loss_e}{2} \tag{8}$$

The joint loss is used to backpropagate the weights across the entire model in order co-train the triple extraction and entity typing components.

[6]https://github.com/huggingface/neuralcoref

# 4 Annotated Datasets

## 4.1 Problem Formulation

There are currently no datasets available for evaluating triple extraction systems due to the lack of a suitable data format for representing a dynamic number of ground truth triples within text. Existing KGC from text systems employ pipelines of named entity recognition, relation extraction, coreference resolution and entity linking separately. There are three primary issues arising from the current data formats used to train KGC from text systems:

1. Complexity: training a KGC from text system requires multiple datasets, each with different formats, i.e. one for NER, one for relation extraction, and so on;
2. The heads and tails of a triple may not necessarily be named entities, and hence applying NER to identify potential heads/tails is bound to miss important concepts that are not named entities; and
3. Relation extraction identifies whether two given named entities are related, but often a concept is related to multiple other concepts rather than one single concept.

We therefore introduce a novel data format that allows for the task of triple extraction to be treated as a multi-label sequence labelling problem. We take inspiration from entity typing research (Ling and Weld 2012; Ren et al. 2016), where the training examples are given as a list of tokens and entity mentions. Figure 5 shows an example of this data format (top), and how we have applied the format to model the data for knowledge graph construction (bottom).

Using this data format a model can learn to predict both triple membership and entity types. For the triples, a model may learn to predict one or more labels per token which represent the membership of that token in a particular triple. These predictions can then be converted to a set of triples by searching for the heads, relations and tails of each triple using a simple algorithm: starting from $n = 1$, find a contiguous sequence of tokens whose predicted labels contain $head\_n$, and repeat for $rel\_n$ and $tail\_n$. These three sets of tokens form the first triple, and the process is repeated until no further triples are found or $n > T$, where $T$ is an upper limit on the number of triples. The ideal value of $T$ depends on the dataset; we found $T = 10$ to be sufficient for our evaluation datasets.

In summary, the problem formulation of our proposed annotation format is as follows: *given a sentence, label each token with both its membership in one or more triples, as well as its corresponding entity type(s).*

```
{tokens:   ["The", "Urus", "is", "rumored", "to", "be", "powered", "by", "a",
            "twin-turbo", "4.0-liter", "V8", "generating", "600", "hp."],
 mentions: [{start: 0,  end: 1,  labels: ["product", "product/car"]}]}
```

```
{tokens:   ["The", "Urus", "is", "rumored", "to", "be", "powered", "by", "a",
            "twin-turbo", "4.0-liter", "V8", "generating", "600", "hp."],
 mentions: { triples: [
             {start: 1,  end: 2,  labels: ["t1", "t1/head"]},
             {start: 6,  end: 8,  labels: ["t1", "t1/rel"]},
             {start: 9,  end: 12, labels: ["t1", "t1/tail", "t2", "t2/head"]},
             {start: 12, end: 13, labels: ["t2", "t2/rel"]},
             {start: 13, end: 15, labels: ["t2", "t2/tail"]}]}]
             entity_types: [
             {start: 1,  end: 2,  labels: ["product", "product/car"]}]}]}
```

Figure 5: The standard annotation scheme for entity typing (top) and its translation to knowledge graph construction (bottom).

| | # Documents | | | Avg Tokens/doc | | | Avg Triples/doc | | | Avg Entities/doc | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Name** | **Train** | **Dev** | **Test** | **Train** | **Dev** | **Test** | **Train** | **Dev** | **Test** | **Train** | **Dev** | **Test** |
| CS | 162 | 20 | 20 | 184.5 | 184.7 | 176.0 | 7.5 | 8.5 | 6.3 | - | - | - |
| AE | 646 | 81 | 81 | 217.2 | 206.3 | 210.1 | 7.3 | 7.3 | 7.6 | - | - | - |
| BBN | 1639 | 183 | 166 | 28.7 | 28.5 | 27.9 | 3.6 | 4.2 | 3.9 | 2.4 | 2.2 | 2.2 |

Table 1: A summary of the three datasets introduced in this paper.

### 4.2 Data Collection & Preprocessing

In this paper we introduce three datasets[7]: *Catering Services* (CS), *Automotive Engineering* (AE), and *BBN*. The *CS* and *AE* datasets contain news articles relating to the catering and automotive industries, respectively. The *BBN* dataset contains single sentences taken from Wall Street Journal articles.

The *CS* and *AE* datasets were obtained by scraping a variety of catering and automotive news websites, respectively, ranging from articles dating 2012-2014. We took the first $n$ paragraphs of each article until the total character count of the scraped document exceeded 800. Articles shorter than 800 characters in length were not included in the dataset. Each article in each dataset was tokenised using NLTK (Bird, Klein, and Loper 2009).

The data collection process was different for the *BBN* dataset, which is a standard benchmarking dataset for entity typing (Ren et al. 2016). We simply took a small portion of the *BBN* dataset, which has already been tokenised and annotated for entity typing by (Ren et al. 2016), and labelled each token with their triple membership(s).

After processing, each dataset was split into 80% training, 10% validation and 10% test data. The detailed statistics of each dataset is displayed in Table 1.

### 4.3 Annotation Procedure

Each dataset was manually annotated using Redcoat (Stewart, Liu, and Cardell-Oliver 2019), a powerful web-based tool for labelling data for hierarchical entity typing. In contrast to other popular annotation tools such as BRAT (Stenetorp et al. 2012) and Amazon SageMaker Ground Truth[8], Redcoat supports multi-label annotation, meaning a single token can be labelled with multiple categories. This was critical to our application as it allowed for many tokens to be the head, relation, and/or tail of multiple triples, as seen in Figure 5 where *4.0-liter V8* is both the tail of the first triple and the head of the second triple.

Annotation took place over several days, with three annotators in total. A meeting was held to ensure consistent annotation between annotators. Several rules were agreed upon during this meeting:

1. A rule of thumb for annotating heads and tails is that they should ideally be concepts appearing in Wikipedia.
2. A rule of thumb for annotation relations is that they should typically be verbs.
3. The heads, relations and tails should be as condensed as possible (e.g. the only head, relation and tail in *Bugatti unveiled their brand new Veyron model* should be *Bugatti*, *unveiled*, and *Veyron*, respectively).
4. The first appearance of a coreferring mention should be labelled as the head of every corresponding triple, e.g. in the sentence *Mercedes is a company. It manufactures cars*, the first triple should be (*Mercedes, is, company*) and the second should be (*Mercedes, manufactures, cars*) as opposed to (*it, manufactures, cars*).
5. Annotators should aim for approximately 6-10 triples per document, with the exception of the *BBN* dataset where annotators should aim for 1-3 triples per document.

[7]The datasets are available on GitHub:
https://github.com/Michael-Stewart-Webdev/Seq2KG

[8]https://aws.amazon.com/sagemaker/groundtruth/

6. If a document is too difficult to label with a high level of confidence it should not labelled and hence not be included in the dataset.

Each dataset was annotated for triple membership using the annotation scheme shown in Figure 5. The BBN dataset, which was already labelled for entity typing by (Ren et al. 2016), is the only dataset that is labelled for entity typing as well as triple membership.

# 5 Experiments

## 5.1 Models

We evaluate the performance of four models:

- Rule-based: our rule-based model, discussed in Section 3.1;
- Filtering: our filtering model, discussed in Section 3.2;
- End-to-End Seq2KG (no ET): our End-to-End Seq2KG model, discussed in Section 3.3. It learns to predict triples from text, but not the entity types of each head and tail; and
- End-to-End Seq2KG (ET): our End-to-End Seq2KG model, which jointly learns to perform both triple extraction and entity typing.

## 5.2 Model Parameters

After parameter tuning we found that the best performance on the development set for both of our models were achieved with a hidden dimension size of 768, and 0.5 dropout prior to the final layer. The models were optimised using ADAM. The batch size was 5 for the filtering model and 10 for the End-to-End Seq2KG model. The filtering model and End-to-End Seq2KG models were trained with a learning rate of 0.00001 and 0.001, and max sequence length of 10 and 100, respectively. Documents were split into sentences when training and evaluating the End-to-End Seq2KG model in order to fit the max sequence length.

## 5.3 Embedding Similarity (ESim)

Existing research in KGC from text evaluates triples using qualitative manual analysis, as a meaningful evaluation metric that compares the quality of predicted triples to the ground truth triples does not yet exist (Mehta, Singhal, and Karlapalem 2019). It is difficult to combine both structural and semantic similarity in one measure, and hence knowledge graph evaluation metrics typically target one or the other. For example, the metric proposed by (You et al. 2018) incorporates maximum mean discrepancy (Gretton et al. 2012) to measure structural similarity, while the method proposed by (Hawes and Kelleher 2004) combines relation-based and feature-based alignment to measure semantic similarity.

In light of the lack of suitable, easily applicable metrics for evaluating graphs produced by KGC systems, we introduce a novel and simple evaluation metric, termed as *embedding similarity* (or *ESim*), to provide a way to quantitatively evaluate the performance of KGC models. The metric works as follows. For each document, construct a Directed Acyclic Graph $P$ from the predicted triples, and $G$ from the ground truth triples. Then, perform $k$ random walks ($k = 10$ in our evaluation) on each graph to obtain two sets $PR$ and $GR$. The token sequences of $PR$ and $GR$ are then embedded. For example, if $PR_1$ is a subgraph containing the nodes $\{Lamborghini, SUV, Urus\}$, and the edges $\{making, Called\}$ connecting nodes 1 to 2 and 2 to 3 respectively, we embed the sequence *"Lamborghini making SUV Called Urus"*. Each embedded sequence $pr_i$, $gr_i$ in $PR$ and $GR$ is compared pairwise using cosine similarity, and averaged to obtain the final score $s$ for that document:

$$\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{ab}}{\|\mathbf{a}\|\|\mathbf{b}\|} = \frac{\sum_{i=1}^{n} \mathbf{a}_i \mathbf{b}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{a}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{b}_i)^2}} \tag{9a}$$

$$s = \frac{\sum_{i=1}^{k} cos(f(PR_i), f(GR_i))}{k} \tag{9b}$$

where $f$ is a function that computes the average word embedding across a sequence of tokens from a subgraph (BERT in our case). The overall score across the entire corpus is the average of all document-level scores. The score captures the semantic meaning behind each of the heads, relations and tails in each triple via embeddings while also capturing graph-level structural information via the random walk mechanism.

## 5.4 Evaluating Entity Typing Performance

To evaluate the ability of the joint model to perform entity typing alongside triple extraction, we use Strict Accuracy, Loose Macro, and Loose Micro score (Ling and Weld 2012). Given $P$ is the set of predicted entity labels across each token of each head and each tail, and $T$ is the ground truth labels of those same tokens:

- **Strict Accuracy**:

$$precision = (\sum_{e \in P \cap T} \delta(\hat{t_e} = t_e))/|P| \tag{10a}$$

$$recall = (\sum_{e \in P \cap T} \delta(\hat{t_e} = t_e))/|T| \tag{10b}$$

- **Loose Macro**:

$$precision = \frac{1}{|P|}(\sum_{e \in P} \frac{|\hat{t_e} \cap t_e|}{|\hat{t_e}|} \tag{11a}$$

$$recall = \frac{1}{|T|}(\sum_{e \in T} \frac{|\hat{t_e} \cap t_e|}{|\hat{t_e}|} \tag{11b}$$

- **Loose Micro**:

$$precision = \frac{\sum_{e \in P} |t_e \cap \hat{t_e}|}{\sum_{e \in P} |\hat{t_e}|} \tag{12a}$$

$$recall = \frac{\sum_{e \in T} |t_e \cap \hat{t_e}|}{\sum_{e \in T} |\hat{t_e}|} \tag{12b}$$

The F1 scores of each metric are then calculated as follows:

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \tag{13}$$

| | CS | AE | BBN |
|---|---|---|---|
| Rule-based | **0.919** | 0.897 | **0.849** |
| Filtering | 0.691 | 0.197 | 0.427 |
| End-to-End Seq2KG (no ET) | 0.879 | **0.902** | 0.797 |
| End-to-End Seq2KG (ET) | - | - | 0.809 |

Table 2: The results of each model's triple extraction performance on the three datasets according to the ESim metric. Each score is the average score of evaluating the same model 3 times.

| | Mi-f1 | Ma-f1 | Strict acc |
|---|---|---|---|
| End-to-End Seq2KG (ET) | 0.588 | 0.654 | 0.654 |

Table 3: The results of the End-to-End Seq2KG (ET) model's entity typing performance on the BBN dataset in terms of F1 Score. Each score is the average score of evaluating the same model 3 times.

# 6 Results

Our results aim to determine the following:

- **Triple extraction performance**: How well do the filtering and End-to-End Seq2KG models perform triple extraction from text when compared to the rule-based system?
- **Entity typing effectiveness**: How well does the End-to-End Seq2KG model perform entity typing during the process of KGC?

## 6.1 Triple Extraction Performance

Table 2 displays the results of each of the three models on the CS, AE and BBN datasets according to the embedding similarity metric. Note that the End-to-End Seq2KG model was not evaluated on the CS or AE datasets as those datasets are not labelled with entity types.

The results show that while the rule-based system achieves the best overall performance across the three datasets, the End-to-End Seq2KG model performs almost as well in every case. The strong performance of the rule-based system is not surprising given it was designed to extract triples from news datasets. The fact that our End-to-End Seq2KG model performs similarly indicates that End-to-End Seq2KG triple extraction is extremely useful considering it does not require the construction of handcrafted rules to be applied to specific domains.

The filtering model does not perform as well, however. It performs very poorly across all three datasets, especially on the automotive engineering news. There key reason for this phenomenon lies in its preliminary regular-expression based triple extraction method that builds the set of candidate triples. During training, the ground truth triples are appended to the set of candidate triples and are assigned a label of 1, indicating that they are correct. During evaluation, however, the ground truth triples are not known and the model must rely on the greedy triple extraction method to provide it with the correct triples alongside the incorrect ones. In practice it is exceedingly challenging to identify the correct triples using regular expression parsing, and so the model ends up predicting very few triples, hence receiving a low score. We attempted to use OpenIE instead, but it performed even worse due to the excessively long head and tail phrases.

## 6.2 Entity Typing Performance

Table 3 displays the results of the End-to-End Seq2KG (ET) model on entity typing according to F1 Score. For reference, the system proposed by (Ren et al. 2016) achieves a micro, macro and strict F1 score of 0.638, 0.698 and 0.710 respectively on the full BBN test dataset. Their model is trained on the entire corpus, however, which contains nearly 30,000 documents in contrast to our small subset of 1639 documents.

The results show that the model is certainly capable of learning to perform entity typing as part of the KGC process, although its performance could be improved. A notable finding is that learning to perform entity typing does not negatively affect the triple extraction performance, which suggests that it would be possible to also learn to predict other attributes directly from the text as well.

# 7 Conclusion

In this research, for the first time, in the same flavour as Collobert et al.'s seminal work of "Natural language processing (almost) from scratch" in 2011, we proposed a Seq2KG model attempting to achieve *"Knowledge graph construction (almost) from scratch"*. Through an end-to-end trainable model, Seq2KG jointly learns to generate triples and resolves entity types as a multi-label classification task through deep learning neural networks. The model training is enabled through a new annotation scheme we devised, and trained and tested on three quality annotated datasets. To ensure a fair and strict evaluation, we proposed a new evaluation that captures both graph-level structural information as well as word-level semantic information. The Seq2KG model outperforms a neural filtering model and on par with a state-of-the-art prize winning rule-based model which is domain specific with heavily hand-crafted rules.

As the first work of its kind, the paper presented here can be seen as a framework with substitutable components. Other binary classifiers can be used in the filtering model, such as support vector machines. Different methods can be applied to the joint loss function in the end-to-end model, or the sequence labelling component could be replaced with other sequence labelling models such as the transformer (Vaswani et al. 2017). It would also be interesting to test the ability of the model to predict structured relations as opposed to verb-based relations from the text. More research into evaluation metrics is also needed; knowledge graph isomorphism or ontology mapping would be potential areas of interest.

# References

Allahyari, M.; Pouriyeh, S.; Assefi, M.; Safaei, S.; Trippe, E. D.; Gutierrez, J. B.; and Kochut, K. 2017. A brief survey of text mining: Classification, clustering and extraction techniques. *arXiv preprint arXiv:1707.02919*.

Bird, S.; Klein, E.; and Loper, E. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".

Chen, P.; Lu, Y.; Zheng, V. W.; Chen, X.; and Li, X. 2018. An automatic knowledge graph construction system for k-12 education. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, 40. ACM.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Choudhury, S.; Agarwal, K.; Purohit, S.; Zhang, B.; Pirrung, M.; Smith, W.; and Thomas, M. 2017. Nous: Construction and querying of dynamic knowledge graphs. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, 1563–1565. IEEE.

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

Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; and Kuksa, P. 2011. Natural language processing (almost) from scratch. *Journal of machine learning research* 12(Aug):2493–2537.

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

Gretton, A.; Borgwardt, K. M.; Rasch, M. J.; Schölkopf, B.; and Smola, A. 2012. A kernel two-sample test. *Journal of Machine Learning Research* 13(Mar):723–773.

Hawes, N., and Kelleher, J. 2004. Analogy hy alignment: On structure mapping and similarity. In *STAIRS 2004: Proceedings of the Second Starting AI Researchers' Symposium*, volume 109, 205. IOS Press.

Kertkeidkachorn, N., and Ichise, R. 2017. T2kg: An end-to-end system for creating knowledge graph from unstructured text. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*.

Koehn, P. 2009. *Statistical machine translation*. Cambridge University Press.

Ling, X., and Weld, D. S. 2012. Fine-grained entity recognition. In *AAAI*.

Martinez-Rodriguez, J. L.; Lopez-Arevalo, I.; and Rios-Alvarado, A. B. 2018. Openie-based approach for knowledge graph construction from text. *Expert Systems with Applications* 113:339–355.

Mehta, A.; Singhal, A.; and Karlapalem, K. 2019. Scalable knowledge graph construction over text using deep learning based predicate mapping. In *Companion Proceedings of The 2019 World Wide Web Conference*, 705–713. ACM.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Ren, X.; He, W.; Qu, M.; Huang, L.; Ji, H.; and Han, J. 2016. Afet: Automatic fine-grained entity typing by hierarchical partial-label embedding. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 1369–1378.

Schmitz, M.; Bart, R.; Soderland, S.; Etzioni, O.; et al. 2012. Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 523–534. Association for Computational Linguistics.

Stenetorp, P.; Pyysalo, S.; Topić, G.; Ohta, T.; Ananiadou, S.; and Tsujii, J. 2012. Brat: a web-based tool for nlp-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, 102–107. Association for Computational Linguistics.

Stewart, M.; Liu, W.; Cardell-Oliver, R.; and Griffin, M. 2017. An interactive web-based toolset for knowledge discovery from short text log data. In *International Conference on Advanced Data Mining and Applications*, 853–858. Springer.

Stewart, M.; Enkhsaikhan, M.; and Liu, W. 2019. Icdm 2019 knowledge graph contest: Team uwa. In *Proceedings of the 2019 IEEE International Conference on Data Mining (ICDM)*, 1546–1551. IEEE.

Stewart, M.; Liu, W.; and Cardell-Oliver, R. 2019. Redcoat: A collaborative annotation tool for hierarchical entity typing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, 193–198.

Vasile, D., and Lukasiewicz, T. 2018. *Learning Structured Video Descriptions: Automated Video Knowledge Extraction for Video Understanding Tasks: Confederated International Conferences: CoopIS, CTC, and ODBASE 2018, Valletta, Malta, October 22-26, 2018, Proceedings, Part II*. 315–332.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Wang, C.; Ma, X.; Chen, J.; and Chen, J. 2018. Information extraction and knowledge graph construction from geoscience literature. *Computers & geosciences* 112:112–120.

Wu, X.; Wu, J.; Fu, X.; Li, J.; Zhou, P.; and Jiang, X. 2019. Automatic knowledge graph construction: A report on the 2019 icdm/icbk contest. In *2019 IEEE International Conference on Data Mining (ICDM)*, 1540–1545. IEEE.

You, J.; Ying, R.; Ren, X.; Hamilton, W. L.; and Leskovec, J. 2018. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*.

Zhou, Y.; Sun, Y.; and Honavar, V. 2019. Improving image captioning by leveraging knowledge graphs. 283–293.