Natural Language Engineering (2019), **25**, pp. 219–238
doi:10.1017/S1351324918000451

CAMBRIDGE UNIVERSITY PRESS

ARTICLE

Neural architectures for open-type relation argument extraction

Benjamin Roth[1,*], Costanza Conforti[2], Nina Poerner[1], Sanjeev Kumar Karn[1] and Hinrich Schütze[1]

[1]Center for Information and Language Processing, Ludwig Maximilian University of Munich, München, Germany and
[2]Language Technology Laboratory, University of Cambridge, Cambridge, UK
*Corresponding author. Email: beroth@cis.uni-muenchen.de

(Received 27 November 2017; revised 15 October 2018; accepted 15 October 2018; first published online 7 December 2018)

Abstract

In this work, we focus on the task of *open-type relation argument extraction (ORAE)*: given a corpus, a query entity *Q*, and a knowledge base relation (e.g., "*Q* authored notable work with title *X*"), the model has to extract an argument of non-standard entity type (entities that cannot be extracted by a standard named entity tagger, for example, *X*: the title of a book or a work of art) from the corpus. We develop and compare a wide range of neural models for this task yielding large improvements over a strong baseline obtained with a neural question answering system. The impact of different sentence encoding architectures and answer extraction methods is systematically compared. An encoder based on gated recurrent units combined with a conditional random fields tagger yields the best results. We release a data set to train and evaluate ORAE, based on Wikidata and obtained by distant supervision.

Keywords: information extraction; machine learning; question answering; text data mining

1. Introduction

Systems for turning unstructured information from textual corpora (such as Wikipedia and newspaper corpora) into structured representations are crucial tools for harnessing the vast amounts of data available online. Automatic detection of relations in text allows humans to search and find relevant facts about entities, and it allows for further processing and aggregation of relational information. A prototypical user for such a system would be, for example, an analyst who is interested in facts about a specific organization or person, or a social scientist who is interested in aggregating facts over time for trend detection.

Entity-driven relation extraction is the problem of identifying relevant facts for a query entity *Q* (e.g., *Q* = "Steve Jackson") in a large corpus according to a pre-defined relational schema that defines relations such as "*Q* authored notable work with title *X*". Systems solving this task are often complex pipelines containing modules for information retrieval, linguistic pre-processing, and relation classification (cf. Surdeanu 2013).

While the main focus in relation extraction has previously been on *relation classification* (i.e., predicting whether a relation holds between two given arguments), quantitative analysis has repeatedly shown that *argument identification* (often performed by carefully engineered submodules) has at least as big of an impact on end-to-end results (Roth 2015). Moreover, in previous benchmarks (Surdeanu 2013; Zhang *et al.* 2017), relations have been selected such that the vast majority of arguments are of standard types (e.g., *person*, *location*, *organization*) and can be detected by a named entity recognizer. Even for standard named entity types,

argument identification is hard for complex cases like nested named entities because different levels of granularity are relevant for different relations.

Consider, for example, the following two named entity tagging errors:

- *[Popular Kabul]$_{ORG}$ lawmaker [Ramazan Bashardost]$_{PER}$, who camps out in a tent near parliament ...*
- *[Haig]$_{PER}$ attended the [US Army]$_{ORG}$ academy at [West Point]$_{LOC}$ ...*

In the above example, a pipelined system which relies on the tagging output cannot extract *Kabul* as the *city-of-residence* for the query *Ramazan Bashardost*. It cannot also extract *US army academy at West Point* as the *school-attended* for the query *Haig*, even though the relation is expressed explicitly by the verb *attended*. Argument identification of non-standard types (e.g., a title of a book or a work of art), which is the focus of this work, is even more challenging.

Comparison of end-to-end relation extraction systems, as in the knowledge base population (KBP) English Slot Filling shared task (Surdeanu 2013; Angeli *et al.* 2014), indicates that recall is the most difficult metric to optimize in entity-driven relation extraction. Further analysis (Pink *et al.* 2014) showed that named entity tagging is, after relation prediction, the main bottleneck accounting for roughly 30% of the missing recall. It is also worth noting that tagging or matching errors may harm twice: once for missing the correct answer and secondly for returning an incorrect answer span.

The key motivation for our research is that *identification of the query entity* is relatively easy and causes few errors: string match and expansion heuristics using information retrieval methods work well and need not rely on entity tagging. In contrast, *identification of the slot filler* is hard, especially if a diverse range of entity types is considered. Consequently, *we give the relation prediction model full freedom to select a slot filler from all possible sub-sequences of retrieved query contexts.*

Based on this motivation, we define the task of *open-type relation argument extraction (ORAE)*, a more general form of entity-driven relation classification. In contrast to the standard setting (which has been the focus of KBP), the key novelty of ORAE is that slot fillers of any type are admissible; they are not restricted to the standard entity types like *person* and *location*. Broadening the definition of types at the same time allows us to broaden the definition of relations, and we can handle relations that pose difficulty for standard relation classification.

Most slot filling methods make heavy use of named entity recognition (Zhang *et al.* 2016), but named entity recognizers address only pre-defined types (for which there is training data with annotated entities). Non-standard types cannot be recognized without special engineering (e.g., compiling lists of entities or writing regular expressions). To address this, we propose a set of new relation argument extraction methods in this article that do not require a named entity recognizer.

In summary, this article makes the following contributions:

- The formulation and motivation of ORAE as a problem in information extraction, and a novel data set for Wikidata relations that contain an argument that is of non-standard type.
- A range of different neural network architectures for solving ORAE and their evaluation in extensive experiments:
  - We compare different neural architectures (*encoders*) for computing a sentence representation suitable for argument extraction. The proposed encoders are based on convolutional networks (Collobert *et al.* 2011), recurrent networks (Chung *et al.* 2014), and self-attention (Vaswani *et al.* 2017).
  - We compare different neural architectures (*extractors*) for extracting answers from this sentence representation. The proposed extractors are based on pointer models (Vinyals *et al.* 2015), linear chain conditional random fields (Lafferty *et al.* 2001; Lample *et al.* 2016), and table filling (Miwa and Sasaki 2014).

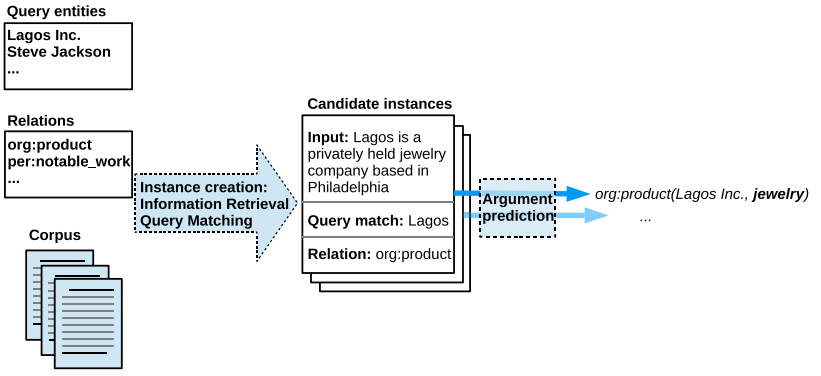

Figure 1. Schematic overview of a query-driven KBP system. The focus of this work is developing an argument prediction component that can extract non-standard entities.

2. Encoding and extraction architectures

A big class of errors in end-to-end relation extraction systems are missing or inexact named entity tags and, in a pipelined model, lost recall cannot be regained (Pink *et al.* 2014; Roth 2015). The models we propose aim at overcoming this problem by skipping the named entity recognition step altogether and instead predicting a slot filler (or none) for query entities and the relations of interest. Our models do not perform a separate task of entity recognition; but of course they have to do entity recognition implicitly since extracting a correct slot filler requires correct assessment of its type and correct assessment of the type of the query entity. The aim of this work is to develop models that predict knowledge graph relations for concepts that have non-standard type in a query-driven setup and to explore a wide range of possible solutions to this problem.

Figure 1 shows the general setup in which our argument prediction models can be applied. The practical scenario is one where a user seeks to extract relational information from a large text corpus for a list of relevant query entities and relations (depending on the query entity type, Surdeanu 2013). We call this scenario query-driven KBP. In query-driven KBP, input to the argument prediction model is a context that has been provided by the retrieval system for the relevant query entity, for example:

- **Query:** *"Alexander Haig"*
- **Context:** *"Haig attended the US army academy at Westpoint."*

The relation of interest is also provided to the model (if there are several possible relations for a query type, several instances are created). In traditional approaches to query-driven KBP, the query is matched by the retrieval system (often using expansion heuristics) and a second potential argument is marked by named-entity tagging, and a simple classification prediction has to be made for all potential relations, for example:

- *"[Haig]$_{Query}$ attended the [US army]$_{Answer}$ academy at Westpoint."* `works-for` ⇒ Yes/No?
- *"[Haig]$_{Query}$ attended the [US army]$_{Answer}$ academy at Westpoint."* `school-attended` ⇒ Yes/No?
- *"[Haig]$_{Query}$ attended the US army academy at [Westpoint]$_{Answer}$."* `born-in` ⇒ Yes/No?
- ...

In our ORAE approach, the answer has to be identified simultaneously with deciding whether the relation holds or not.

- *"[Haig]$_{Query}$ attended the US army academy at Westpoint."*
  `works-for` ⇒ Answer?
  `school-attended` ⇒ Answer?
  `born-in` ⇒ Answer?
  ...

We conceptually break our models for argument prediction down into three components:

- **Lookup layer:** Representation of the context sentence. We use the same input representation throughout our experiments.
- **Encoder:** Layers that compute a representation for every position in the sentence, combining information from other positions.
- **Extractor:** Last part of the architecture; it computes the extracted answer as the output.

A *model* consists of the lookup layer followed by an encoder layer, followed by a decoder layer. The remainder of this section provides a detailed discussion of layer variants.

2.1 Lookup layer

In our problem formulation (argument extraction), a query entity and relation of interest are provided to the model, and the missing argument has to be found. The model is therefore conditioned on the query, and it has knowledge of the query position. We indicate the query position through wildcarding, where we replace the query by a special token `<QUERY>`, and additionally we also use *position embeddings* to indicate the distance of other tokens to the query position. The relation in question is already provided at this stage to the model through the learned *relation embeddings*. There is one embedding per relation.

Specifically, the lookup layer provides embeddings for five types of information useful for answer extraction that are concatenated for each position in the input context (see Figure 2). For input position i, the input representation vector $\mathbf{e}_i$ is a concatenation of vectors[a]:

$$\mathbf{e}_i = [\mathbf{e}(w_i); \mathbf{e}(p_i); \mathbf{e}(s_i); \mathbf{e}(i-j); \mathbf{e}(r)] \quad (1)$$

- **Word embeddings** (embedding size 100). Words contained in the pretrained GloVe vectors[b] are initialized with those vectors, otherwise they are initialized randomly. The vector $\mathbf{e}(w_i)$ is the embedding of w_i, the word at position i.
- **Affix embeddings.** Prefix and suffix embeddings (length: 2 characters, embedding size: 100) are learned in order to capture simple part-of-speech or named entity-type generalization patterns (capitalization, morphological indicators). The vectors $\mathbf{e}(p_i)$ and $\mathbf{e}(s_i)$ are the embeddings of the prefix and suffix of w_i, respectively.
- **Position embedding.** Since the first experiments using convolutional neural networks (CNNs) for relation extraction (Collobert *et al.* 2011; dos Santos *et al.* 2015) encoding the relative position to relation arguments has been key to good performance. We encode the relative position with respect to the query. Position encoding is used for all extractors, not only CNNs. The vector $\mathbf{e}(i-j)$ is the embedding of the relative position $(i-j)$ w.r.t. the query position (j). The position embedding has size 10.

[a]Vectors are column vectors by default. Semicolons $[\cdots;\cdots]$ indicate vertical stacking along the column-axis, and commas $[\cdots,\cdots]$ indicate horizontal concatenation along the row-axis.

[b]https://nlp.stanford.edu/projects/glove/

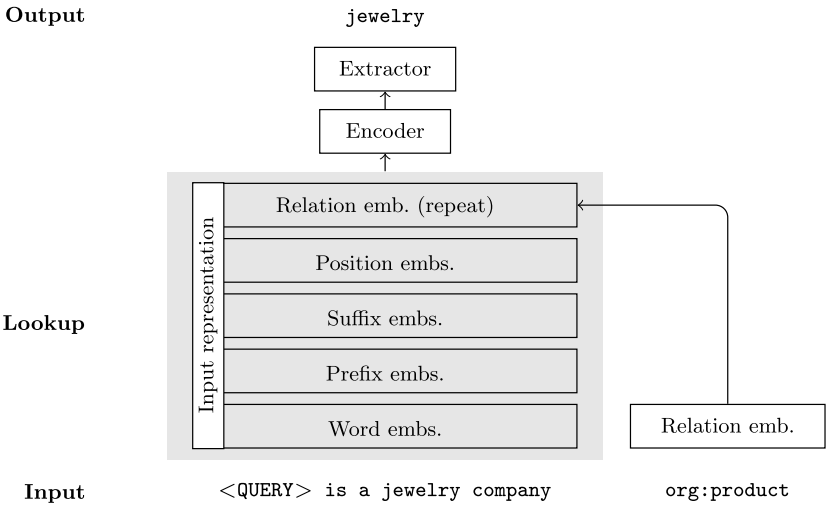

Figure 2. Lookup layer and architecture overview.

- The **relation embedding** identifies the relation to the model and is repeated for every position in the input context. The vector $\mathbf{e}(r)$ is the embedding of the relation r. The relation embedding size is set to 12, the number of relations.

We denote the dimensionality of the input representation as k ($k = 3 * 100 + 10 + 12 = 322$). All embedding vectors are fine-tuned during training. The $k \times n$ matrix containing the input representations for all n positions is denoted by $E = [\mathbf{e}_1, \ldots, \mathbf{e}_n]$.

2.2 Encoders

The sentence encoder translates the output of the lookup layer with neural network architectures that consider a wider context. We use three different alternative instantiations.

2.2.1 Recurrent neural network (RNN) encoder

In the RNN encoder architecture, each candidate sentence is encoded by two layers of bi-directional Gated Recurrent Units (GRUs) (Chung *et al.* 2014) with a hidden size of 200 (100 per direction). The hidden representation for position i in the first GRU layer is the concatenation of a left-to-right and a right-to-left GRU hidden state. It is denoted by:

$$\mathbf{h}_i^{(1)} = [\overrightarrow{\mathbf{h}}_i^{(1)}; \overleftarrow{\mathbf{h}}_i^{(1)}] \tag{2}$$

Where the GRU hidden states are computed via the recurrences:

$$\overrightarrow{\mathbf{h}}_i^{(1)} = \mathrm{GRU}(\overrightarrow{\mathbf{h}}_{i-1}^{(1)}, \mathbf{e}_i) \tag{3}$$

$$\overleftarrow{\mathbf{h}}_i^{(1)} = \mathrm{GRU}(\overleftarrow{\mathbf{h}}_{i+1}^{(1)}, \mathbf{e}_i) \tag{4}$$

The second layer GRU takes the first layer as input and computes $\mathbf{h}_i^{(2)} = [\overrightarrow{\mathbf{h}}_i^{(2)}; \overleftarrow{\mathbf{h}}_i^{(2)}]$ accordingly:

$$\overrightarrow{\mathbf{h}}_i^{(2)} = \mathrm{GRU}(\overrightarrow{\mathbf{h}}_{i-1}^{(2)}, \mathbf{h}_i^{(1)}) \tag{5}$$

$$\overleftarrow{\mathbf{h}}_i^{(2)} = \mathrm{GRU}(\overleftarrow{\mathbf{h}}_{i+1}^{(2)}, \mathbf{h}_i^{(1)}) \tag{6}$$

We did not observe a significant increase in performance on development data when using more layers, so the encoder output for the RNN encoder is $\mathbf{h}_i^{\text{RNN}} = \mathbf{h}_i^{(2)}$.

2.2.2 CNN encoder

CNNs are used with padding such that the number of input steps equals the number of output steps. We use 4 different filter widths: 3, 5, 7, and 9. For each filter width, we stack 3 layers with 32, 64, and 128 filters, respectively. The ReLU activation is applied to each filter, and dropout (drop probability of 0.2) is applied between the convolutional layers. The outputs of the last layer (for each filter width), and the relation embedding, are concatenated and used as input for the answer extractor.

More specifically, for filter width 3, the first layer CNN computes a 32-dimensional representation vector $\mathbf{h}_i^{(1;3)}$ (we write $\mathbf{h}_i^{(1;x)}$ for filter width x) where each entry $\mathbf{h}_{i,f}^{(1;3)}$ is computed from the input representation using the $3 * k$-dimensional weight vector $\mathbf{w}_f^{(1;3)}$ for a particular filter f, and the ReLU activation[c]:

$$\mathbf{h}_{i,f}^{(1;3)} = \text{ReLU}(\mathbf{w}_f^{(1;3)T}\mathbf{e}_{[i-1:i+1]}) \tag{7}$$

where $\mathbf{e}_{[i-1:i+1]} = [\mathbf{e}_{i-1};\mathbf{e}_i;\mathbf{e}_{i+1}]$ and ReLU is defined component-wise as $\text{ReLU}(\mathbf{x}) = \max(\mathbf{0}, \mathbf{x})$.

The second (and third) layer CNN computes a representation of size 64 (and 128) using the analogous formula:

$$\mathbf{h}_{i,f}^{(2;3)} = \text{ReLU}(\mathbf{w}_f^{(2;3)T}\mathbf{h}_{[i-1:i+1]}^{(1;3)}) \tag{8}$$

(Respectively $\mathbf{h}_{i,f}^{(3;3)} = \max(0, \mathbf{w}_f^{(3;3)T}\mathbf{h}_{[i-1:i+1]}^{(2;3)})$ for the final third layer.)

The analogous formulas are applied for filter widths 5, 7, and 9 (only considering wider contexts $[i-2{:}i+2]$, *etc.*). The final output of the CNN encoder is the concatenation of the third layer output for each filter width. For the CNN architecture (but not for the other encoders), we observed small improvements on the development data by again concatenating the relation embeddings at each position:

$$\mathbf{h}_i^{CNN} = [\mathbf{h}_i^{(3;3)};\mathbf{h}_i^{(3;5)};\mathbf{h}_i^{(3;7)};\mathbf{h}_i^{(3;9)};\mathbf{e}(r)] \tag{9}$$

2.2.3 Self-attention encoder

A third encoder uses the multi-headed self-attention architecture of Vaswani *et al.* (2017) to get an encoding for each position in the sequence. In self-attention, the input representation for each position is used as a query to compute attention scores for all positions in the sequence. Those scores are then used to compute the weighted average of the input representations.

In multi-headed self-attention, input representations are first linearly mapped to lower-dimensional spaces, and the output vectors of several attention mechanisms (called *heads*) are concatenated and form the output of one multi-headed self-attention layer. An attention head a encodes a sequence of input vectors into a sequence of output vectors $\mathbf{h}_i^{(a)}$. Different heads pay attention to (i.e., put weight on) different parts or interactions in the input sequence. Different heads are parametrized independently (the respective parameters are marked by a superscript (a) to indicate that they are head-specific).

For one attention head a in the first self-attention layer, we obtain the vector for position i:

$$\mathbf{h}_i^{(a)} = \text{Attention}\,(W^{q(a)}\mathbf{e}_i, W^{K(a)}E, W^{V(a)}E) \tag{10}$$

[c]We omit the bias term in affine transformations for readability.

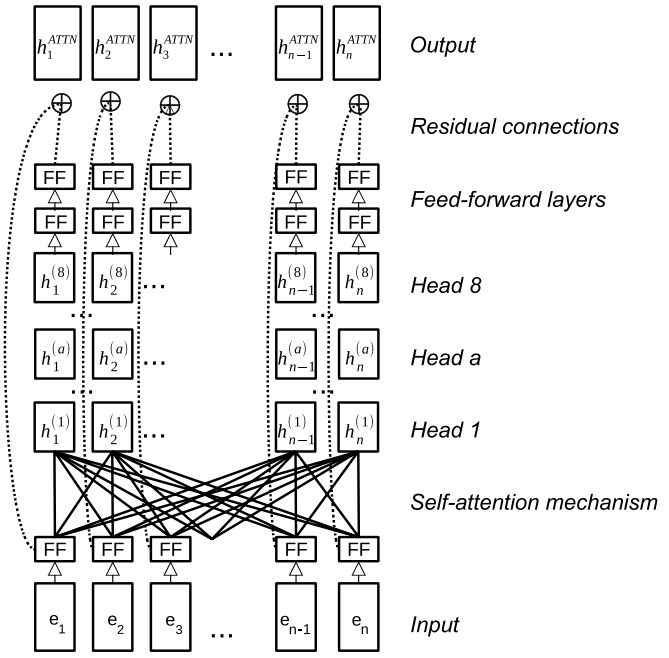

Figure 3. Schematic diagram of self-attention layer.

where $W^{q(a)}$, $W^{K(a)}$, and $W^{V(a)}$ are linear transformations (matrices specific to head a) to map the input representation into lower-dimensional space, and the matrix $E = [\mathbf{e}_1, \ldots, \mathbf{e}_n]$ is the matrix that contains the input representation (e.g., from the lookup layer, Section 2.1). The function computing the resulting vector (from $\mathbf{q} = W^{q(a)}\mathbf{e}_i$, $K = W^{K(a)}E$, and $V = W^{V(a)}E$) is defined by:

$$\text{Attention}\,(\mathbf{q}, K, V) = V\,\text{softmax}\,(K^T\mathbf{q}) \tag{11}$$

We follow the setup described in Vaswani *et al.* (2017) and use 8 attention heads (each with a hidden size of 25 resulting in an overall hidden size of 200). The input to the self-attention mechanism is transformed by a feed-forward layer (output size 200, ReLU activation), and the output of the attention heads at each position is followed by two feed-forward layers (output sizes 400 and 200, ReLU activations). One self-attention layer (the combination of self-attention heads and feed-forward layers) is stacked three times. More repetitions did not yield significant improvements on development data. See Figure 3 for a diagram depicting the architecture of one self-attention layer.

We deviated from the setup described in Vaswani *et al.* (2017) in the following ways, each of which improved the performance on the development data:

1. We included residual connections that add the input of the self-attention mechanism directly to the output, rather than having two residual connections within each layer.
2. We used batch normalization (Ioffe and Szegedy 2015) rather than layer normalization (Ba *et al.* 2016) after concatenation of the attention heads and the MLP, respectively.

As for the RNN and CNN encoders, the result is a vector representation $\mathbf{h}_i^{\text{ATTN}}$ for each position in the sentence.

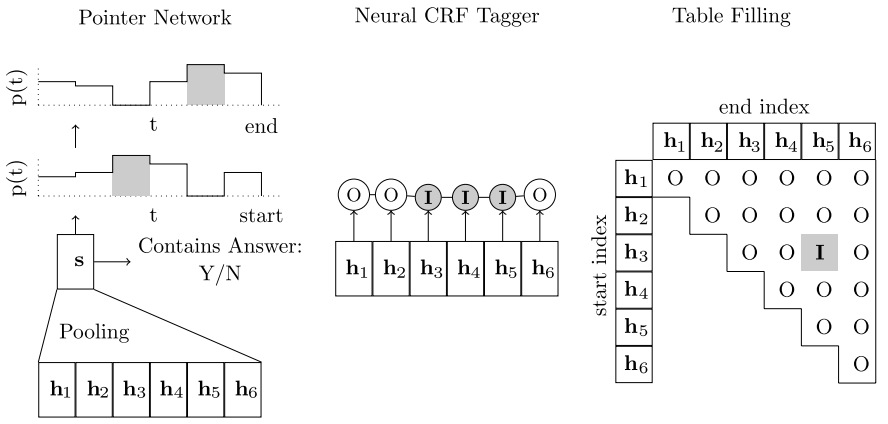

Figure 4. Three extractor frameworks described in Section 2.3 for predicting an answer span (from position 3 to 5 in this example, indicated by gray shading) from the encoder outputs $\mathbf{h}_i$.

2.3 Extractors

Extractors take the encoder output and predict the argument span (conditioned on the query entity and the relation of interest). If there is no argument for the relation of interest, the empty span is returned. We use three different architectures for argument extraction. In the following, the encoder output at position i is denoted by $\mathbf{h}_i$, irrespective of whether it stems from the RNN, CNN, or self-attention encoder. The matrix H represents the encoder outputs for all positions in the sentence, and its dimensionality is length of the sentence times encoder output size.

2.3.1 Pointer network

Pointer networks (Vinyals *et al.* 2015) are a simple method to point to positions in a sequence by calculating scores (similar to attention), normalizing them using softmax and taking the argmax. In our case, two pointers are predicted, pointing to the start and end positions of the relation argument.

Figure 4 (left third) shows the processing flow for the pointer network. First, a *summary vector* $\bar{\mathbf{s}}$ is computed for the whole sentence by max-pooling over the sentence encoder representation (output of "Encoder" in Figure 2) and applying a fully connected layer with `ReLU` activation:

$$\bar{\mathbf{s}} = \text{ReLU}(W^s \text{Pool}(H)) \tag{12}$$

where Pool returns a vector containing the row-wise maximum of a matrix and W^s is a learned affine transformation.

A binary **label** is predicted through logistic regression from the summary vector $\bar{\mathbf{s}}$; this label indicates whether the sentence contains an answer argument or not. The summary vector $\bar{\mathbf{s}}$ is also used as a context vector to compute the pointer scores for predicting the **start position**, in a way similar to attention modeling. For each position in the sentence, the summary vector is concatenated with the encoder output representation $\mathbf{h}_i$ at this position, and from this a score is predicted (using an MLP with one hidden layer of size 200) indicating how strongly this position should be associated with the start of a relevant argument. The softmax gives a distribution over the start positions:

$$p(start = i) = \text{softmax}\,(\text{MLP}([\bar{\mathbf{s}};\mathbf{h}_i])) \tag{13}$$

The **end position** is predicted by the same mechanism, but in this case the context vector is not the summary vector $\bar{\mathbf{s}}$. Instead, the softmax distribution over start positions output by the

previous step is used as the context vector (and concatenated with the encoder outputs $\mathbf{h}_i$ for score prediction). For sentences that do not contain an answer argument, the start and end positions are set to point to the query entity position during training. This way we hope to bias predictions to be closer to the query entity position. At test time we exclude any predictions where either the probability that an answer is contained is less than or equal to 0.5, or where the span overlaps with the query entity position.

2.3.2 Conditional random field (CRF) tagger

The CRF tagger model predicts the answer span by predicting the label `"I"` for the answer, and `"O"` otherwise. As in previous work combining neural networks with CRFs (Collobert *et al.* 2011; Lample *et al.* 2016), the CRF combines local label scores, obtained from the features of the previous layers, with learned transition weights in order to obtain sequential label consistency: for an entire label sequence $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ the global score is defined as:

$$s(H, \mathbf{y}) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} s_{i,y_i} \tag{14}$$

where A is a (learned) matrix of transition scores from label y_i to label y_{i+1} (a special start label y_0 is assumed), and s_{i,y_i} is the local label score for label y_i, obtained by a (learned) linear mapping from $\mathbf{h}_i$.

Viterbi decoding is used to find the predicted answer spans. The local label scores are also used in our system to assign a confidence value and to find the most likely answer span if there are several predicted spans.

2.3.3 Table filling

The table filling extractor jointly looks at pairs of sentence positions and decides for each pair whether they are start and end positions for the query and relation on which the network is conditioned. For a start position i and an end position j, the table filling model decides whether those positions describe the start and end of the sought answer. The table filling model uses the encoder outputs $\mathbf{h}_i$, $\mathbf{h}_j$ as the input for this binary decision (`I`: subspan is answer, `O`: subspan is not an answer, see Figure 4, right diagram).

Compared to the pointer network (three model outputs: label, start, end) and the CRF tagger (number of model outputs = length of sequence), the table filling model has the most number of outputs to predict, as it needs, in principle, to pair each position in a sentence with all other (subsequent) positions in the sentence. To reduce the amount of computation that follows from this quadratic complexity, we limit the maximum length of representable answers to be 5 (which covers 98% of actually occurring answers). Note that—even though we exclude a large number of "negative" cells from the table and do not do any prediction for them—the vast majority of output cells still has the negative label (all but one pair of positions is not a relevant relation argument), introducing a strong bias which may make it harder for the model to predict a positive label at all. For the combination with the CNN encoder, it was necessary to double the weights for the positive class, following Gülçehre *et al.* (2016), to deal with the highly skewed distribution of output classes (otherwise the table filling model would predict no answers).

For each pairing (i, j) of potential start and end positions, we concatenate the encoder vectors for the two positions and predict the corresponding cell value of the table. Logistic regression is used for cell prediction:

$$p(is\_answer = True | start = i, end = j) = \sigma([\mathbf{h}_i; \mathbf{h}_j]^T \mathbf{w}^{(table)}) \tag{15}$$

where a different weight vector $\mathbf{w}^{(table)}$ is learned for each answer length.

We experimented with deeper architectures for cell value prediction, but did not observe any improvements, presumably due to the overwhelming majority of cells with a negative label.

2.4 Hyperparameters

The following hyperparameters were tuned on the development data (according to instance-level accuracy) using random search (Bengio 2012) over the ranges given below. For tuning, the encoders were paired with the pointer network extractor (which is most similar to the Bidirectional Attention Flow (BiDAF) baseline, Section 4.2.1). We did not tune any hyperparameters specific to the extractors.

- learning rate: {0.1, 0.01, 0.001}
- number of CNN/GRU/Self Attention layers: {1, 2, 3, 4}
- CNN, maximal window size: {5, 7, 9}
- CNN, maximal number of filters: {64, 128, 256}
- Self-attention, output size[d] (=number of heads * head size): {50, 100, 200, 400}
- GRU, hidden size: {50, 100, 200, 400}

The resulting hyperparameter choices are reported in the sections describing the respective submodels. We use the 100-dimensional pretrained GloVe vectors of Pennington *et al.* (2014) and did not experiment with other word vector variants. The size of the relation vector is equal to the number of relations (12, as for one-hot-encoding, but with the flexibility to arrange similar relations closer to each other in embedding space). We found that for the position embedding size a value equal to the square root of the maximum relative distance (in our experiments 10) gave good performance, and increasing it further did neither improve nor hurt the model. All models use the Adam optimizer, and the best value for learning rate was 0.01 for all models. We found that larger batch sizes in general yielded better results than smaller ones, resulting in a batch size of 512 (which was the largest we could efficiently process on our infrastructure).

3. Data set

The models for predicting knowledge graph relations between entities that have non-standard type, proposed in the previous sections, are evaluated using a distantly supervised data set that we extracted from Wikidata and Wikipedia specifically for this purpose.

We first identify relations that meet three specific criteria and retrieve entity pairs for these relations. The three criteria are the following:

(a) **Non-standard type.** We look for relations that have one argument of a standard type, the *query*, and one argument of a non-standard type, the *slot*. Training and evaluation are done for the task of identifying correct fillers for the slot. We consider the MUC-7 named entity types (location, person, organization, money, percent, date, and time) as standard types (Chinchor and Robinson 1997).

(b) **Open class.** There must be a wide range of admissible values for the slot in question (i.e., the answers must be *relational*, not *categorical* (Hewlett *et al.* 2016). For example, the Wikidata relation P21 (*sex or gender*) has a non-standard argument slot, but only a handful of distinct possible values are attested in Wikidata; so, P21 is not a relation that we consider for our

[d]Following Vaswani *et al.* (2017), we use eight heads.

Table 1. The table gives, for each relation, its name and an example sentence. (Query entity and correct answer entity are indicated in brackets)

Relation	Example sentence
`per:occupation`	[*Alan Aubry*]$_Q$ (born 24 September 1974) is a French [*photographer*]$_A$.
`per:position_held`	Under pressure , former [*Tánaiste*]$_A$ [*Erskine H. Childers*]$_Q$ agreed to run.
`per:conflict`	It is named for [*Henry Knox*]$_Q$, an [*American Revolutionary War*]$_A$ general.
`per:notable_work`	In the [*Steve Jackson*]$_Q$ Games card game [*Munchkin*]$_A$, there is a card called "Dwarf Tossing".
`per:participant_of`	[*Ahlm*]$_Q$ was listed among the top ten goalscorers at the [*2008 Olympics*]$_A$ tournament.
`per:award_received`	[*Alex Smith*]$_Q$'s name was put on the [*Stanley Cup*]$_A$ in 1927 with Ottawa.
`per:field_of_work`	While teaching at Berkeley, [*John Harsanyi*]$_Q$ did extensive research in [*game theory*]$_A$.
`org:industry`	Select stores offer [*fast food*]$_A$ outlets such as [*Subway*]$_Q$ and Taco Bell.
`per:noble_family`	Stefan was the son of Lazar and his wife [*Milica*]$_Q$, a lateral line of [*Nemanjić*]$_A$.
`per:ethnic_group`	[*Hamdi Ulukaya*]$_Q$ was born in 1972 to a [*Kurdish*]$_A$ family in Turkey.
`org:product`	[*Lagos*]$_Q$ is a privately held American [*jewelry*]$_A$ company
`gpe:office`	Brown was the de facto [*premier*]$_A$ of [*Province of Canada*]$_Q$ in 1858.

data set. As a threshold, we require Wikidata to contain at least 1000 distinct values for the slot in question.

(c) **Substantial coverage.** There must be a large number of facts (argument pairs) in Wikidata for a relation to be eligible for inclusion in our data set. We require the minimal number of facts to be 10,000 for each relation.

We check criterion (a) using the Wikidata relation descriptions. We use the Wikidata query interface[e] and the SPARQL query language to check criteria (b) and (c), and to retrieve entity pairs (and their surface forms) for all relations. The relation entity pair tuples are randomly split into training (50%), development (25%), and test data (25%).

In a second step, we retrieve sentences containing argument pairs (*distant supervision sentences*). An English Wikipedia dump (2016-09-20) is indexed on the sentence level using ElasticSearch.[f] For each relation argument, aliases are obtained using Wikipedia anchor text and the query expansion mechanism of the RelationFactory KBP system (Roth *et al.* 2014). Up to 10 sentences are retrieved for each argument pair. Although criterion (c) requires a minimum number of 10,000 facts, we are not able to find a distant supervision sentence for every pair. Therefore, the number of actually occurring facts is less than 10,000 for some relations. For each positive instance (sentence-relation-argument tuple), we sample two negative instances by replacing the relation with different relations (uniformly chosen at random).

Examples of resulting positive instances for each relation are shown in Table 1. Table 2 gives an overview of the training, development, and test data sizes. Table 3 lists the relations, together with the number of training sentences for each relation. We renamed the Wikidata ids to be more readable (similar to TAC KBP relations[g]): the names contain the entity type of the query argument as a prefix.[h]

[e] https://query.wikidata.org/
[f] https://www.elastic.co/
[g] https://tac.nist.gov/about/index.html
[h] The data set and code are released at http://cistern.cis.lmu.de/orae/.

Table 2. Number of instances, positive instances, distinct fact triples, and distinct query entities for training, development, and test data

	Train.	Dev.	Test
No. of instances	673.677	340.050	335.883
No. of positive instances	224.559	113.350	111.961
No. of fact triples	132.983	66.925	66.697
No. of query entities	89.349	46.967	47.155

Table 3. The table gives for each relation its name, Wikidata id, and number of training instances

Relation	id	No. of sentences
`per:occupation`	P106	57,693
`per:position_held`	P39	47,386
`per:conflict`	P607	20,575
`per:notable_work`	P800	18,826
`per:participant_of`	P1344	14,646
`per:award_received`	P166	13,330
`per:field_of_work`	P101	13,059
`org:industry`	P452	12,352
`per:noble_family`	P53	9260
`per:ethnic_group`	P172	7169
`org:product`	P1056	6482
`gpe:office`	P1313	3781

4. Experiments

4.1 Evaluation setup

Each encoder architecture is combined with every extractor architecture. We compute accuracy, precision, and recall by assessing exact string match. We compute accuracy on a per-instance (query–relation–context) level. For precision, recall, and f-measure, two variants are computed: instance level and tuple level.

In the instance-level setup, the items which are considered are combinations of *query–relation–context–answer* (where *context* is a particular sentence represented by its id, and *answer* is the missing argument that is to be extracted). In the tuple-level evaluation, the sentence id is ignored, and the same fact-tuple is counted only once, even if it has been extracted from several sentences, that is, the items to be considered are combinations of *query–relation–answer*. The tuple-level evaluation measures how well the ground-truth *facts* are recovered, that is, it corresponds to the quality of a knowledge graph obtained with the extraction algorithm, since repeated extractions are only counted once.

Precision and recall are computed from the sets of items, where relevant = set(correct items) and retrieved = set(predicted items); the f-measure is computed as usual $f = 2pr/(p + r)$.

$$p = \frac{|\text{relevant} \cap \text{retrieved}|}{|\text{retrieved}|}$$

$$r = \frac{|\text{relevant} \cap \text{retrieved}|}{|\text{relevant}|}$$

4.2 Baselines

4.2.1 Argument extraction using BiDAF

Levy *et al.* (2017) formulate relation extraction as a reading comprehension problem: for each relation, a set of natural language questions is written by humans, and answers are extracted using the BiDAF network (Seo *et al.* 2016). In one of their experiments (the *KB Relations* setting), they do not provide the full questions, but rather give the relation as an unanalyzed atom (the question corresponds to the relation as the only pseudo-word). This setting is applicable to our problem definition (and is simultaneously their best performing setup), hence we choose this system as a baseline. Since Levy *et al.* (2017) adapted a question answering (QA) model to the task of relation answer extraction, some parts of the model setup that help for analyzing natural language questions (such as the attention mechanism that aligns parts of the sentence with parts of the question) are superfluous and not helpful for our task. A number of elements of BiDAF are similar to our model, but instantiated in a different way. (i) Seo *et al.* (2016) use character embeddings, and we use prefix and suffix embeddings. (ii) In BiDAF, attention is driven by the query. In one of our settings, we use self-attention where any input information (words or relation) can recombine information from the whole sentence. (iii) Similar to the prediction of start and end points in Seo *et al.* (2016), one of our architectures is a pointer network. We compare this to two other design choices for predicting the answer span.

4.2.2 Relation classification using Positional Attention

We also compare to the Position-aware Attention (PosAtt) model of Zhang *et al.* (2017), a strong relation classifier that can be used in a pipelined setting. The PosAtt model requires as input a sentence with the *query* and an already identified (by sequence tagging or string matching) *answer candidate*. PosAtt encodes this input with a neural architecture that summarizes the sentence using an attention mechanism that is aware of query and answer candidate positions, and predicts a relation for the encoded sentence.

Since the relational arguments in ORAE are of non-standard type and cannot be detected by off-the-shelf named entity taggers, we identify answer candidates by string matching: potential answers for a relation are all substrings in a sentence that were arguments for that relation in the training data.

4.3 Results and analysis

4.3.1 Architecture comparison

Table 4 compares all combinations of encoder and extractor architectures introduced in the previous sections. In order to keep the overview uncluttered, we only show tuple-level results in Table 4. See Table 5 for additional instance-level results for selected architectures.

Encoders. For the encoder architectures, one can see that the self-attention mechanism (ATTN) is the weakest (although competitive to the baselines, see below), reaching an f-measure of 75.89 in the best combination.[i]

[i]Unless indicated otherwise, we discuss tuple-level scores.

Table 4. Each cell contains P/R/F on tuple level. The best values for each encoder (i.e., per row) are underlined, and the best values for each extractor (i.e., per column) are marked in bold

	Pointer network			Neural CRF tagger			Table filling		
	P	R	F	P	R	F	P	R	F
ATTN	75.50	73.51	74.49	72.39	76.41	74.35	78.11	73.78	75.89
CNN	78.23	**80.62**	**79.41**	**82.59**	76.84	79.61	**78.47**	79.76	79.11
RNN	**78.80**	79.17	78.99	82.53	**81.19**	**81.86**	77.92	**81.44**	**79.64**

Table 5. Comparison with Levy et al. BiDAF model, and Zhang et al. PosAtt model applied to our task. Reported results are P/R/F on tuple level and P/R/F/Acc on instance level. Best results per column marked in bold

	Tuple level			Instance level			
	P	R	F	P	R	F	Acc
BiDAF	70.86	78.76	74.60	76.35	75.84	76.10	90.11
PosAtt	**83.65**	72.11	77.45	–	–	–	–
CNN/CRF	82.59	76.84	79.61	**86.25**	73.48	79.35	90.02
RNN/Table	77.92	**81.44**	79.64	82.31	**78.38**	80.30	91.03
RNN/CRF	82.53	81.19	**81.86**	86.20	78.25	**82.03**	**91.55**

Good results are obtained by the CNN encoder, with the f-measure reaching 79.61 (and with similar results obtained when different extractors are chosen). A slightly higher f-measure of 81.86 is achieved with the RNN encoder; however, for this encoder, results vary more depending on the choice of extractor.

Compared to RNN and CNN, **self-attention modeling** is the *least local* of all three encoders, as it can incorporate information from the entire sentence by the same mechanism; positional information is only captured via the positional embeddings. The comparatively weak performance of the ATTN encoder indicates that some locality bias may be beneficial for argument extraction (higher influence of neighboring words, distance to query), and that non-local modeling, only relying on positional embeddings, is not sufficient.

The **CNN encoder** is the *most local* of all encoders: information of neighboring words is combined using the stacked filters. The only long-range dependency that can be captured is the distance to the query (via positional embeddings). The relatively good results of the CNN encoder indicate that most relevant information can be captured by this mechanism.

The **RNN encoder** can use *all non-local information* via its bidirectional recurrences, but at the same time RNNs have a bias towards local information as it needs to go through fewer transformations. In our experiments this way of encoding the entire sentence information via RNNs yields the best results overall.

Extractors. The **pointer network** is for none of the encoders the best extractor. However, differences to the other extractors are relatively small. The limitation of the pointer network is that decisions for start and end position are not optimized jointly (the score distribution over end positions cannot influence that over start positions), and this fact may limit the model to gain the last percentage points of extra performance needed.

The **neural CRF tagger** is the best extractor for both the CNN and the RNN encoder, achieving the best results overall. Start and end positions are jointly modeled and globally optimized via the tag sequence and the transition scores.

The **table filling extractor** models start and end positions jointly by design. The biggest difficulty for the table filling extractor is the fact that the number of negative labels (combinations

Table 6. Performance of full input representation minus performance of reduced input representation, for Self-attention+Table Filling, CNN+CRF Tagger, RNN+CRF Tagger. Ablated elements: word embeddings, affix embeddings, position embeddings, query wildcarding, and relation embeddings

Architecture	Word	Affix	Position	Query	Relation
ATTN+Table	0.16	1.73	3.54	4.89	50.76
CNN+CRF	2.63	0.16	0.16	2.87	74.06
RNN+CRF	2.73	0.05	−0.29	3.71	79.99

of start and end positions that do not constitute a correct answer) grows quadratically with the sentence length. Without correcting for this imbalance by doubly weighting positive labels in the objective function, recall values would be extremely low—for the CNN encoder without this reweighting no answer would be extracted at all. Despite its relatively good performance, the table filling extractor is therefore less stable than the pointer network or CRF extractor.

Lookup layer. We include an ablation analysis, to examine how different input representations interact with encoder layers and end-to-end models. For each encoder architecture, we take its best combination with a decoder and compare its performance using the full input representation and its performance with a reduced input representation (in terms of tuple-level f-measure), and we report this difference in Table 6. We ablate word embeddings, affix embeddings, and position embeddings, and we compare to a setup where the query is not wildcarded. We also compare to a setup where the relation of interests was not given to the model (i.e., the model loses the capability to distinguish between different relations).

The CNN and RNN models rely more on **word embeddings**, while the self-attention model relies more on **affix embeddings**. **Position embeddings** are crucial for the self-attention encoder (Vaswani *et al.* 2017), in contrast, CNN and RNN model sequential order by design and do not depend on position embeddings. **Query wildcarding** is the most important factor in representing the input. Without query wildcarding, the model may be prone to overfit the queries seen during training, and moreover the information about what element in the sentence is the query is passed on to the model only via the relative position embeddings. Not surprisingly, **relation embeddings** are essential to the performance of the models.

4.3.2 Baselines

Table 5 shows the performance of the BiDAF architecture adapted to relation extraction as in Levy *et al.* (2017) upon training and testing on the ORAE task. We provide a full comparison (precision, recall, f-measure, accuracy, and instance and tuple level) of this baseline to our best-performing (by tuple-level f-measure) encoder–extractor architectures (RNN/CRF, CNN/CRF, and RNN/Table).

The number of instances considered in PosAtt differs from that in answer extraction models (BiDAF and our approaches), since for one query and relation (an instance in answer extraction) there can be many or no answer candidates. We therefore only consider tuple-level scores for comparison with PosAtt. PosAtt does not have the freedom to predict any substring as an answer since it depends on answer candidate identification as a preceding step in a pipeline. It consequently has the lowest recall of all considered models. The good precision of PosAtt indicates, however, that it is a very strong relation classification model.

As for uninformed baselines (like NER-based pipelined systems that cannot detect non-standard types), always predicting the empty answer would yield an accuracy of 66.67%. For the f-measure, there is no simple uninformed baseline, so the base score for the f-measure would be close to 0. Hence, all models perform quite well on the task, extracting answers with accuracies of $\sim$ 90%.

Clearly, our best-performing Neural CRF Tagger has approximately +7% absolute better f-measure in both instance- and tuple-wise evaluation than the conceptually similar BiDAF model. We attribute the improvements of most of our encoder–extractor-based models to the following design choices:

- We wildcard the query entity (`<QUERY>` in Figure 2). This directs extractors to focus their search for the slot filler on the vicinity of the query. Since most answers occur close to the query, introducing this bias improves performance. Wildcarding also prevents overfitting since the model cannot learn from the specific lexical material of the query.
- The combination of prefix and suffix embeddings is advantageous because most of the information about possible non-standard entity types that is not already captured by word embeddings is captured by these two affixes.
- BiDAF devotes modeling capacity to *bidirectional attention* (in order to detect relevant parts of a question), which is irrelevant in the relation scenario since the "question" is represented as exactly one token, that is, the relation itself.
- CRF and table filling answer extraction can model start and end positions jointly, while BiDAF predicts them independently.

To summarize, our experiments indicate that for relation argument extraction, an RNN network with a tagging-based answer extractor is superior to extractors based on table filling or based on the prediction of start and end positions (as often done by QA systems such as BiDAF).

4.3.3 Discussion

We have extended and redefined the problem of slot filling to the task of ORAE. The type of model we have proposed to address ORAE is not just a model that solves relation classification (or slot filling); it also jointly solves the task of finding the entities.

There are several advantages to this extension and redefinition of slot filling.

- In ORAE, the model can use all available information in the sentence and optimize decision thresholds for the task at hand (i.e., filler identification), avoiding tagging errors that it cannot recover from.
- In ORAE, the model can be trained by distant supervision. As long as there are surface strings of entity pairs from a knowledge base, the model can be trained. The cooccurrence requirement for two entities during training also provides some disambiguation and filtering of spurious matches.
- Our definition of ORAE treats standard and non-standard named entity types in completely the same way. This enables us to detect non-standard slot fillers like *job titles*, *products,* and *industries* that approaches based on named entity tagging have difficulties with.

One shortcoming of the setup we presented in this article is that only one answer is predicted per query instance. Although the model architecture can easily be reformulated for a more general setting, the problem lies with the sparse distant supervision training data that only rarely contains matches with multiple answers within a given context. Given this lack of training data, it is not clear how the parameters of such a more general model should best be estimated.

5. Related work

In opinion recognition, early work has focused on extracting opinion holders and opinion items with CRFs and integer linear programming (Choi *et al.* 2006). See Culotta *et al.* (2006) and Hoffmann *et al.* (2010) for other approaches to argument tagging using traditional feature-based CRFs. This line of research has recently been extended (Katiyar and Cardie 2016) to a neural

tagging scheme, where relations (and the distance to the related token) are predicted per token by a long short-term memory network (LSTM, Hochreiter and Schmidhuber 1997). This setting is quite different from ours since prediction is not conditioned on a query entity; apart from the different problem formulation, this also implies that the model cannot be trained with incomplete annotation via distant supervision (Mintz *et al.* 2009), since training needs all labels to be present (not just those for the query Q). Zheng *et al.* (2017) use a tagging scheme similar to Katiyar and Cardie (2016) to annotate relation arguments in sentences. They do not condition on a query entity and need to downweight non-argument labels to overcome sparsity in the training data.

Similarly, table filling models have been developed to extract entities and relations, see Miwa and Sasaki (2014) for the original feature-based formulation and Miwa and Bansal (2016) for an RNN-based extension of the model. In contrast to our work, this model requires fully annotated data (no distant supervision) and therefore has only been applied to relations with standard named entities (*person, location, organization*), where the motivation for open-type argument extraction is less strong. Another extension (Gupta *et al.* 2016) obtained improvements by relying on already identified named entity spans. We compare a variant of neural table filling that does not rely on any of these conditions with a range of alternative argument extraction methods.

Wikireading (Hewlett *et al.* 2016) is the task of extracting infobox properties from Wikipedia articles about a certain entity (similar to Hoffmann *et al.* 2010). Some aspects of Wikireading are easier than the problem we are dealing with, for example, it is guaranteed that there is an answer for every paragraph in the data set, and the query entity is guaranteed to be the topic of the article. Other aspects are more difficult, for example, only 46% of the answers in the data set are contained as exact strings, the majority has to be inferred. In contrast, we are concerned with the problem of predicting whether relations hold between mentions as they are expressed in text.

Another approach to overcoming reliance on named entity recognition in relation extraction is to do segmentation of text heuristically based on part-of-speech patterns and cooccurrences, and then to proceed in the traditional instance-based paradigm (Ren *et al.* 2017).

Traditional relation classification and, more generally, work deciding whether a relation holds between two identified subparts of a sentence is also relevant. Collobert *et al.* (2011) combined CNNs with position embeddings and CRFs for semantic role labeling. Subsequent work confirmed that CNNs are appropriate models for relation classification (Zeng *et al.* 2014; dos Santos *et al.* 2015; Adel *et al.* 2016; Vu *et al.* 2016). Other approaches have employed RNN variants for representing sentences for relation classification (Verga *et al.* 2016; Xu *et al.* 2016).

Another related field is that of QA. The introduction of the Stanford Question Answering Data set (Rajpurkar *et al.* 2016) has given rise to a large body of work on answer extraction. Seo *et al.* (2016) and Chen *et al.* (2017) introduce an efficient method of aligning question and paragraph words through an attention mechanism (Bahdanau *et al.* 2014) to obtain an answer span. Wang *et al.* (2017) propose an architecture that, based on matching LSTM, builds a question-aware passage representation and uses an attention-based pointer network (Vinyals *et al.* 2015) to predict the start and end positions of the answer.

Recently, Levy *et al.* (2017) presented an approach that bridges QA and query-driven answer extraction. They convert the traditional entity-driven relation extraction to a QA setup by crowdsourcing knowledge base relations into natural language questions. They utilize the BiDAF networks of Seo *et al.* (2016) to extract answers. We compare our experimental results to this strong baseline.

6. Conclusion

We have defined the task of ORAE, where the model has to extract relation arguments without being able to rely on an entity extractor to find the argument candidates. ORAE can be viewed as a type of entity-driven slot filling, the task of identifying and gathering relational information about a query entity from a large corpus of text. However, the most common approaches to slot filling

are pipelined architectures, in which relation classification is an isolated step that heavily relies on pre-processing modules such as named entity recognition, to which a large part of end-to-end errors can be attributed. Our approach to ORAE has two conceptual advantages. First, it is more general than slot filling as it is also applicable to non-standard named entity types that could not be dealt with previously. Second, while the problem we define is more difficult than standard slot filling, we eliminate an important source of errors: tagging errors that propagate throughout the pipeline and that are notoriously hard to correct downstream.

We have presented a distantly supervised data set for training and evaluating ORAE models, based on Wikidata relations; the arguments in our data set are non-standard-type named entities, for example, notable work (which can be any title of a book or other work of art) or product (which can be any product name).

We have experimented with a wide range of neural network architectures to solve ORAE, each consisting of a *sentence encoder*, which computes a vector representation for every sentence position, and an *argument extractor*, which extracts the relation argument from that representation. We experimented with CNNs, RNNs, and self-attention as sentence encoders, and with pointer network, conditional random fields tagging, and table filling as argument extractors. Every encoder was combined with every extractor, and high accuracy was obtained for most combinations. The combination of RNN encoder with conditional random field extractor gave the best results, +4% absolute f-measure better than a state-of-the-art relation classification model based on argument matching, and +7% absolute f-measure better than a previously proposed adaptation of a QA model.

References

Adel H., Roth B. and Schütze H. (2016). Comparing convolutional neural networks to traditional models for slot filling. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, San Diego, California, USA, pp. 828–838.

Angeli G., Tibshirani J., Wu J. and Manning C.D. (2014). Combining distant and partial supervision for relation extraction. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, pp. 1556–1567.

Ba L.J., Kiros R. and Hinton G.E. (2016). Layer normalization. *CoRR*, arXiv:1607.06450.

Bahdanau D., Cho K. and Bengio Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, arXiv:1409.0473.

Bengio Y. (2012). Practical recommendations for gradient-based training of deep architectures. In Montavon G., Orr G.B. and Müller K.R. (eds), *Neural Networks: Tricks of the Trade*, Vol. 7700. Springer, Berlin, Heidelberg, pp. 437–478.

Chen D., Fisch A., Weston J. and Bordes A. (2017). Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vancouver, Canada, pp. 1870–1879.

Chinchor N. and Robinson P. (1997). Muc-7 named entity task definition. In *Proceedings of the 7th Conference on Message Understanding*. http://anthology.aclweb.org/M/M98/.

Choi Y., Breck E. and Cardie C. (2006). Joint extraction of entities and relations for opinion recognition. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Sydney, Australia, pp. 431–439.

Chung J., Gülçehre Ç., Cho K. and Bengio Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, arXiv:1412.3555.

Collobert R., Weston J., Bottou L., Karlen M., Kavukcuoglu K. and Kuksa P.P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research* **12**, 2493–2537.

Culotta A., McCallum A. and Betz J. (2006). Integrating probabilistic extraction models and data mining to discover relations and patterns in text. In *NAACL HLT 2006, Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, New York, USA, pp. 296–303.

dos Santos C.N., Xiang B. and Zhou B. (2015). Classifying relations by ranking with convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, Volume 1, Beijing, China, pp. 626–634.

Gülçehre Ç., Ahn S., Nallapati R., Zhou B. and Bengio Y. (2016). Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Volume 1, Berlin, Germany, pp. 140–149.

Gupta P., Schütze H. and Andrassy B. (2016). Table filling multi-task recurrent neural network for joint entity and relation extraction. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, Osaka, Japan, pp. 2537–2547.

Hewlett D., Lacoste A., Jones L., Polosukhin I., Fandrianto A., Han J., Kelcey M. and Berthelot D. (2016). Wikireading: A novel large-scale language understanding task over Wikipedia. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Berlin, Germany, pp. 1535–1545.

Hochreiter S. and Schmidhuber J. (1997). Long short-term memory. *Neural computation* **9**(8), 1735–1780.

Hoffmann R., Zhang C. and Weld D.S. (2010). Learning 5000 relational extractors. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Uppsala, Sweden, pp. 286–295.

Ioffe S. and Szegedy C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, Volume 37, Lille, France, pp. 448–456.

Katiyar A. and Cardie C. (2016). Investigating LSTMs for joint extraction of opinion entities and relations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Berlin, Germany, pp. 919–929.

Lafferty J.D., McCallum A. and Pereira F.C.N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28–July 1, 2001*, pp. 282–289.

Lample G., Ballesteros M., Subramanian S., Kawakami K. and Dyer C. (2016). Neural architectures for named entity recognition. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, San Diego, California, USA, pp. 260–270.

Levy O., Seo M., Choi E. and Zettlemoyer L. (2017). Zero-shot relation extraction via reading comprehension. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, Vancouver, Canada, pp. 333–342.

Mintz M., Bills S., Snow R. and Jurafsky D. (2009). Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, Volume 2, Suntec, Singapore, pp. 1003–1011.

Miwa M. and Bansal M. (2016). End-to-end relation extraction using LSTMs on sequences and tree structures. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, Berlin, Germany, pp. 1105–1116.

Miwa M. and Sasaki Y. (2014). Modeling joint entity and relation extraction with table representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, pp. 1858–1869.

Pennington J., Socher R. and Manning C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Pink G., Nothman J. and Curran J.R. (2014). Analysing recall loss in named entity slot filling. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014*, Doha, Qatar, pp. 820–830.

Rajpurkar P., Zhang J., Lopyrev K. and Liang P. (2016). Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, Austin, Texas, pp. 2383–2392.

Ren X., Wu Z., He W., Qu M., Voss C.R., Ji H., Abdelzaher T.F. and Han J. (2017). Cotype: Joint extraction of typed entities and relations with knowledge bases. In *Proceedings of the 26th International Conference on World Wide Web*, Perth, Australia, pp. 1015–1024.

Roth B. (2015). *Effective distant supervision for end-to-end knowledge base population systems*. PhD thesis, Saarland University.

Roth B., Barth T., Chrupala G., Gropp M. and Klakow D. (2014). RelationFactory: A fast, modular and effective system for knowledge base population. In *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, Gothenburg, Sweden, pp. 89–92.

Seo M.J., Kembhavi A., Farhadi A. and Hajishirzi H. (2016). Bidirectional attention flow for machine comprehension. *CoRR*, arXiv:1611.01603.

Surdeanu M. (2013). Overview of the TAC2013 knowledge base population evaluation: English slot filling and temporal slot filling. In *Proceedings of the Sixth Text Analysis Conference, TAC 2013*, Gaithersburg, Maryland, USA. https://tac.nist.gov/publications/2013/papers.html.

Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L. and Polosukhin I. (2017). Attention is all you need. *CoRR*, arXiv:1706.03762.

Verga P., Belanger D., Strubell E., Roth B. and McCallum A. (2016). Multilingual relation extraction using compositional universal schema. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, San Diego, California, USA, pp. 886–896.

Vinyals O., Fortunato M. and Jaitly N. (2015). Pointer networks. In Cortes C., Lawrence N.D., Lee D.D., Sugiyama M. and Garnett R. (eds), *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems*, Montreal, Quebec, Canada, pp. 2692–2700.

Vu N.T., Adel H., Gupta P. and Schütze H. (2016). Combining recurrent and convolutional neural networks for relation classification. In *NAACL HLT 2016, The 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, San Diego, California, USA, pp. 534–539.

Wang W., Yang N., Wei F., Chang B. and Zhou M. (2017). Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vancouver, Canada, pp. 189–198.

Xu Y., Jia R., Mou L., Li G., Chen Y., Lu Y. and Jin Z. (2016). Improved relation classification by deep recurrent neural networks with data augmentation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, Osaka, Japan, pp. 1461–1470.

Zeng D., Liu K., Lai S., Zhou G. and Zhao J. (2014). Relation classification via convolutional deep neural network. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, Dublin, Ireland, pp. 2335–2344.

Zhang Y., Chaganty A., Paranjape A., Chen D., Bolton J., Qi P. and Manning C.D. (2016). Stanford at tac kbp 2016: Sealing pipeline leaks and understanding chinese. In *Proceedings of TAC*.

Zhang Y., Zhong V., Chen D., Angeli G. and Manning C.D. (2017). Position-aware attention and supervised data improve slot filling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 35–45.

Zheng S., Wang F., Bao H., Hao Y., Zhou P. and Xu B. (2017). Joint extraction of entities and relations based on a novel tagging scheme. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vancouver, Canada, pp. 1227–1236.

Cite this article: Roth B, Conforti C, Poerner N, Karn SK and Schütze H (2018). Neural architectures for open-type relation argument extraction. *Natural Language Engineering* **25**, 219–238. https://doi.org/10.1017/S1351324918000451