Machine Learning Nanodegree: Capstone Project

Finbar Good

December 15, 2018

1 Definition

1.1 Project Overview

Since Google coined the term "knowledge graph", there has been an increasing interest in the types of systems that fall under this broad term[1]. Despite the interest - or perhaps because of the lure of the name behind it - the term has been applied widely and inconsistently. For the purposes of this proposal, we can settle on this short definition: any store of facts represented as a graph. And the sub-class of these graphs we are specifically interested in are those that make use of the RDF standard.

RDF (Resource Description Framework) graphs [6] represent facts as a triple of IRIs or literal values. A triple consists of three parts:

- a subject: a thing or class, referred to using an IRI
- a predicate: a property or relationship of the subject
- an object: either the value subject's property, or the target of relationship

When the content of a graph is specified with RDF, the subjects and objects form the nodes and the predicates form the edges. Because the predicates are directed - going from subject to object - the graph is directed.

SPARQL is a query language designed for accessing RDF data, and was elevated by the W3C to the recommended language for doing so [8]. The following is an example of a very simple query in SPARQL:

```
SELECT ?creator
WHERE { ?creator imbdo:created imdbr:the_wire }
```

In this example, we are looking for the entity (or entities) that created another entity, imdbr:the_wire. There are superficial similarities with SQL, but they work in very distinct

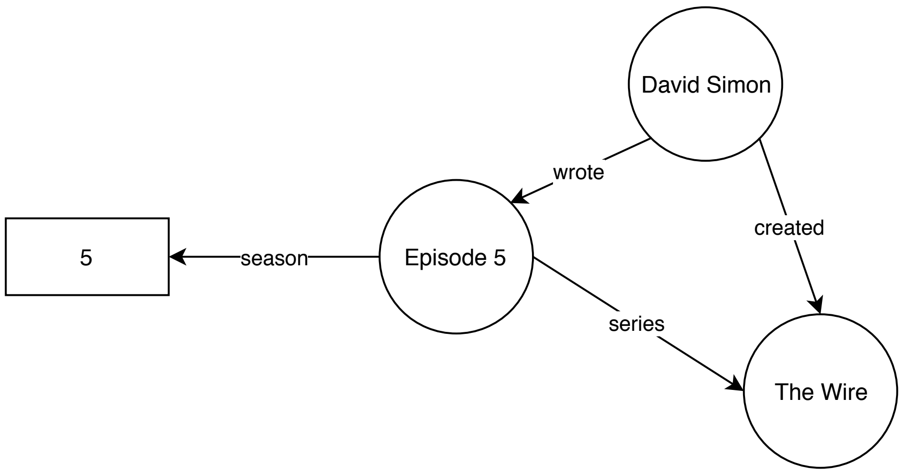

Figure 1: RDF example: some facts about The Wire

ways. What they have in common is, from the point of view of a lay user, a steep learning curve; only specialists tend to have the skills to create queries in either language.

1.2 Problem Statement

An area of research that has emerged in the past decade has been to broaden the accessibility of knowledge graphs through parsing of natural language queries (NLQ) to extract an answer from a knowledge graph (for example [11]). Some of the more recent efforts in the more general topic of question answering with a knowledge base have looked at using neural networks to solve the problem (for example [2]).

The challenge with accessing the data in knowledge graphs using RDF is the query language, SPARQL. For many casual or non-technical users it is an insurmountable barrier. To broaden the reach of such knowledge graphs requires the automated translation of what casual users can produce, natural language queries (NLQ), into something knowledge graphs can parse, SPARQL.

This project aims to show that, at least in principal, a neural network is a viable solution to automated translation of NLQs to SPARQL. Specifically:

1. Download, merge, clean and split the LC-QuAD dataset (see 2.1)
2. Construct a word-oriented encoder-decoder neural network for training the translater
3. Construct a SPARQL inference model that uses the trained encoder-decoder
4. Train the encoder-decoder with the datasets
5. Evaluate effectiveness of the inference model using the BLEU metric

6. Repeat, altering hyper-parameters to investigate drivers of accuracy

The models will be useful for considering the best approach to constructing a real, production-ready NLQ-to-SPARQL translater, supporting search of a knowledge graph. Two key constraints are:

1. Training datasets will, at least initially, be necessarily small
2. The predicted SPARQL must be valid

1.3 Metrics

The BLEU score measures the degree of match between a generated sentence and the reference sentence. This makes it useful to measure success of a translation, in our case from natural language to SPARQL query. Values range from 1 (perfect match) to 0 (perfect mismatch). Roughly, it is a count of the number of distinct n-gram matches found in the reference sentence, normalised by word count.

To use an example given in [4], we have the generated sentence:

```
It is a guide to action which ensures that the military always obeys the
commands of the party
```

If we compare it to this reference sentence:

```
It is a guide to action that ensures that the military will forever heed
Party commands
```

we see that 8 of the bigrams in the generated sentence are found in the reference sentence. The bigram precision score is the number of matches divided by the number of bigrams generated i.e. 8 / 17 = 0.47.

Shorter n-gram scores score precision, longer ones score fluency.

The actual calculation combines a precision measure with a penalty for the generated sentence being shorter than the reference sentence/s, the brevity penalty. The brevity penalty is defined as:

$$BP = \begin{cases} 1 & \text{if } c > r \\ \mathrm{e}^{1-r/c} & \text{if } c \leq r \end{cases}$$

where c is the length of the candidate translation and r the reference corpus length.

The BLEU score is then calculated as:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

where the weighted sum of n-gram precisions p_n, up to N, with weights w_n.

I will use the BLEU metric to compare different training runs on the models. As SPARQL queries are not "fault tolerant", we will need high scores for n > 1.

2 Analysis

2.1 Data Exploration

The LC-QuAD dataset [9] was created for the QALD (Question Answering over Linked Data) initiative [5], a series of annual challenges to translate NLQ into SPARQL (or the correct answer to the NLQ). The LC-QuAD dataset consists of 5000 questions and their corresponding SPARQL queries. The questions were generated using the following workflow:

1. Manually create query templates and a natural language equivalent template
2. Extract a list of entities
3. Manually create a whitelist of predicates
4. For each entity, extract subgraph centred on the entity from DBpedia, extending 2 hops away
5. Generate a query from each fact in these subgraphs, restricted to the predicate whitelist
6. The populated template is then mapped to the natural language equivalent
7. Humans review the final result, paraphrasing and/or correcting the results

Here is an example entry from the dataset:

```
{
  "_id": "285",
  "corrected_question": "In which state is the Channel district?",
  "intermediary_question": "What is the <state> of Channel District ?",
  "sparql_query": " SELECT DISTINCT ?uri WHERE { <http://dbpedia.org/
     resource/Channel_District> <http://dbpedia.org/ontology/state> ?uri }
      ",
  "sparql_template_id": 2
}
```

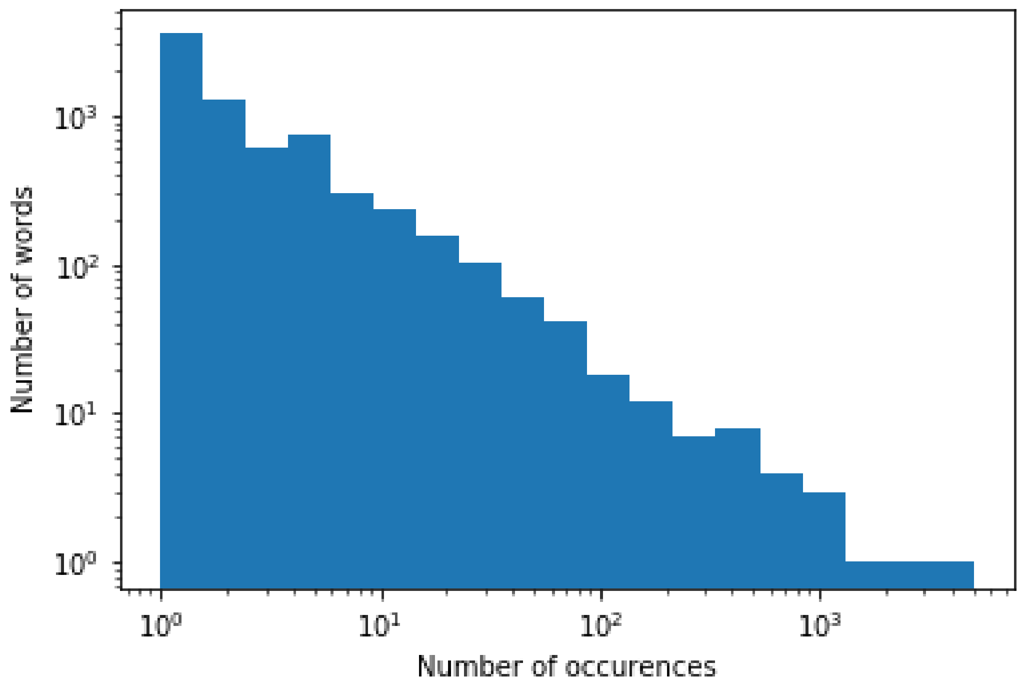

Figure 2: Distribution of words in questions

In this exercise, we are only interested in 2 fields:

- corrected_question: the natural language question. We will normalise these, making their case consistant, removing punctuation etc.
- sparql_query: the target SPARQL query. This will consist of valid SPARQL statements, where each token is well-formed. The only exception is that grouping or continuation symbols aren't forced to have space around them; we will enforce this so they are treated as separate tokens. Otherwise we will not perform any normalization.

2.2 Exploratory Visualization

The word distribution characteristics of the NLQs and SPARQL statements differ in one significant respect: a small but significant number of SPARQL tokens are very frequent.

The distribution for NLQs is shown in figure 2. We can see this follows the expected power law distribution.

The distribution for SPARQL is a little different (see figure 3). There are a significant number of tokens that are not distributed the same as the other tokens. This was mentioned

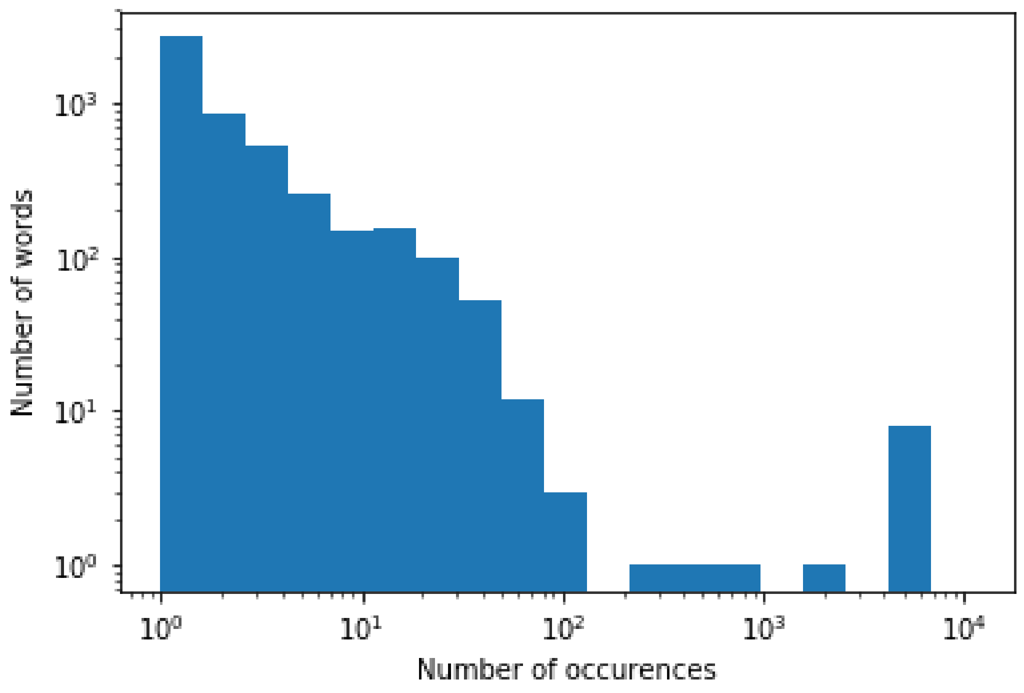

Figure 3: Distribution of words in queries

already above: SPARQL statements are a mixture of a small number of keywords and a large number of identifiers of objects, predicates and literals, the things and relations that the queries are about.

The top 15 SPARQL tokens is given in table 1. In addition to the SPARQL keywords and punctuation, we also see the predicate for type (high in frequncy as narrowing a query by type is so common) as well as a number of classes that may be due to a skewing caused by the method for their generation.

2.3 Algorithms and Techniques

The translation is done with a recurrent neural network (RNN), specifically an encoder-decoder composed of Long Short Term Memory (LSTM) layers. These RNNs have been used to improve the ”memory” of the network, so that the model can be trained to pick up on the larger scale patterns required to translate sentences of a dozen or more words.

I also add an embedding layer as it should improve the computational efficiency of the input questions and queries. However, as the network is being trained to classify - that is to choose the best next token in the sequence - we are required to one-hot encode the predicted query.

Word	**Count**
`?uri`	11010
`.`	6465
`WHERE`	5000
`{`	5000
`}`	5000
`?x`	4740
`SELECT`	4632
`DISTINCT`	4632
`<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>`	1925
`COUNT(?uri)`	658
`ASK`	368
`<http://dbpedia.org/ontology/TelevisionShow>`	236
`<http://dbpedia.org/ontology/Person>`	132
`<http://dbpedia.org/ontology/Film>`	113

Table 1: Top 15 tokens by frequency in the SPARQL queries

The final decoder layer was a dense layer with softmax activation.

I then reused the trained network to generate queries from questions. The question is fed to the encoder, whose LSTM state and a start token are passed to the decoder, to predict the first token of the query. We loop this until the decoder predicts an end token (or hits the maximum query length).

I explored the effects of varying the following:

- Batch size (size of batch between back propagation), to see the effect of speed of learning
- Dropout, to see the degree to which we overfitting is a problem
- Number of neurons, to explore where the sweet spot for over/underfitting might lie
- Optimizer, to compare two that are recommended for LSTMs, Adam and rmsprop

The datasets are shuffled and split into training, validation and test sets.

The training and prediction were done on GPUs.

2.4 Benchmark

I used the model described in [7] as the benchmark. It is also uses a neural network for end-to-end translation of NLQs to SPARQL queries - but is obviously more complex and subjected to many more refinements and iterations, possibly setting a rather unrealistically high bar. Nonetheless it was the only comparable exercise I found that used the BLEU metric.

Depending on the refinement, they achieved BLEU scores between 80.89% and 99.29%. Their dataset was generated directly from DBPedia.

3 Methodology

3.1 Data Preprocessing

The data was downloaded from the GitHub pages of the LC-QuAD project. I then processed the data as follows:

- Consolidated the 2 files, their training and test datasets.
- Loaded, parsed JSON into dict structure, and mapped the ”corrected_question” and ”sparql_query” fields to separate variables.
- The NLQs were cleaned by setting to lowercase and removing punctuation.
- I ensured that the SPARQL queries were properly tokenized, ensuring spaces around grouping and continuation symbols.
- Two datasets were generated from the SPARQL queries: one set had a start token pre-pended, to use as inputs to the decoder; the other had an end token append, to use as outputs.
- All datasets were then tokenized.

3.2 Implementation

Implementation was done with a Jupyter notebook, running on TPUs on Colaboratory.

Following the data pre-processing:

- Sets of hyper-parameters were generated, for (1) number of neurons [64, 128, 256] (2) batch size [32, 64] (3) optimizer ['adam', 'rmsprop'], and (4) dropout [none, 50%]
- For each set, an encoder-decoder model was instantiated, and trained on the training set (split between training-validation-test was 81%-9%-10%). The training was run for 100 epochs (based on observations of the divergence of training and validation

loss, running for any longer wouldn't have been worth it). As this is a classification problem, I used the categorical crossentropy loss function. Although it is not as helpful as BLEU metric for gauging the fluency of translation, I also tracked the accuracy metric.

- A plot of the loss function for training and validation sets was produced at the end of the training run, mainly to gauge overfitting, the biggest problem with a relatively small dataset.
- The training and test datasets were then run through evaluator, which would generate predicted SPARQL queries for each NLQ in the datasets, and both dump the translations (for inspection) and calculate 1-to-4 gram, weighted BLEU scores.

As Colaboratory is a free platform with a time limit for use, the actual iterations through the hyper-parameter sets had to be restarted manually. As a result, the BLEU metrics were consolidated in a selective fashion manually (only picking those that showed the main drivers in improvement to the BLEU scores).

3.3 Refinement

The main challenge that became apparent was the inability of the model to generalize past the most common words, which were the SPARQL keywords and punctuation, plus a few predicates and classes that occur in many queries. You can see the signs of overfitting in figure 4.

I had already implemented a simple grid search of hyper-parameters from the start, but I augmented this with the addition of dropout (leaving it out was an oversight). This greatly improved the translations (although they remained far from useable, as we will see).

4 Results

4.1 Model Evaluation and Validation

Figures 5 and 6 for high-level view of the final training and translating models respectively.

The final model/s were the result of my attempt to port the ideas in a character-based translater described in a blog on the Keras site, to a word-based version.

The model's selling point was that it was able to handle sequences of arbitrary length, something necessary for the problem of arbitrary length questions. Although the final translations themselves were disappointingly far from useable, the fact that after training on a relatively small dataset (for a neural network) they were able to start predicting some

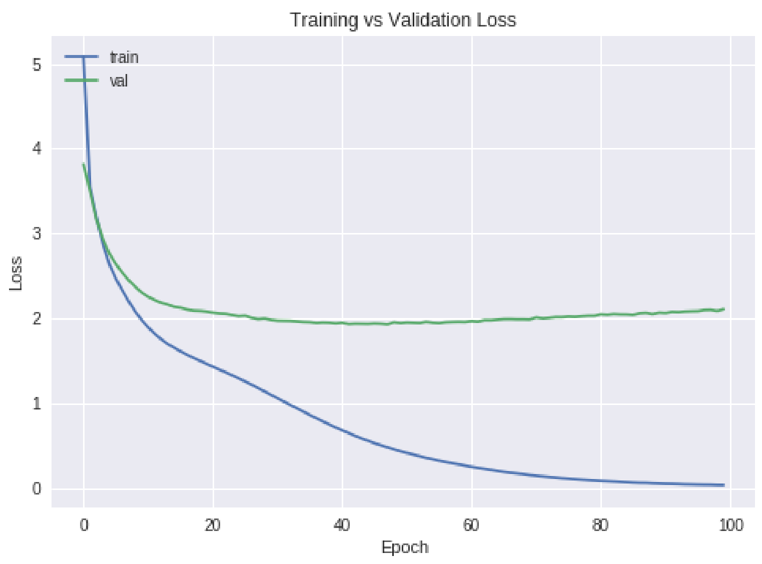

Figure 4: Overfitting evident in the loss function

of the features of a well formed query (for example, balanced parentheses), was impressive, and a vindication of exploring end-to-end translation with a neural network.

Using grid search and with BLEU metric as the measure, the following were the best hyper-parameters:

1. neurons = 256 - lower neurons couldn't learn enough
2. batch size = 64 - although very little variation with this
3. optimiser = rmsprop - the difference with Adam was small
4. dropout = 0.5 - this had the biggest effect

The BLEU scores were robust to repeated runs (which used random shuffles of the source dataset to compose training-validation-test). However, the metric failed to flag runs where the training fell into a sub-optimal minima: even with the above hyper-parameters, the training could fall in on just predicting the most common tokens over and over, and do that for *all* input sentences.

4.2 Justification

A comparison of the BLEU scores is shown in figure 7. We see for the 4-gram BLEU score (evenly weighted) a value exceeding 70%, which when taking in to account the simplicity

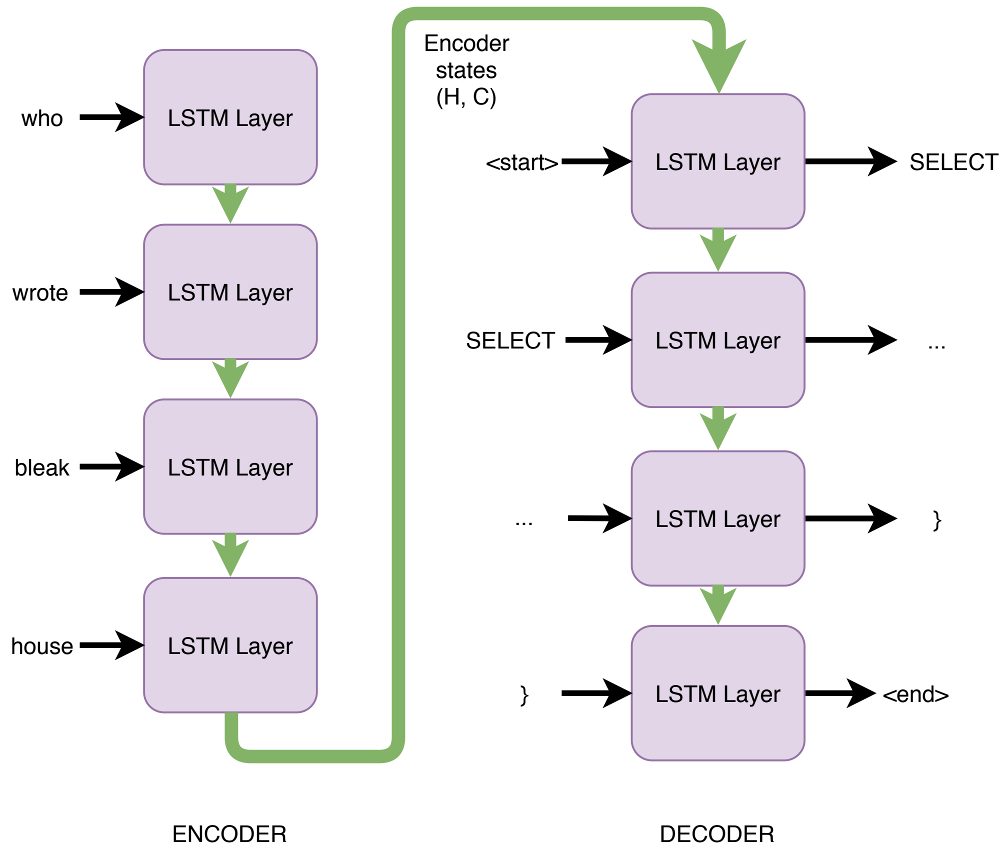

Figure 5: Encoder-Decoder model used in training

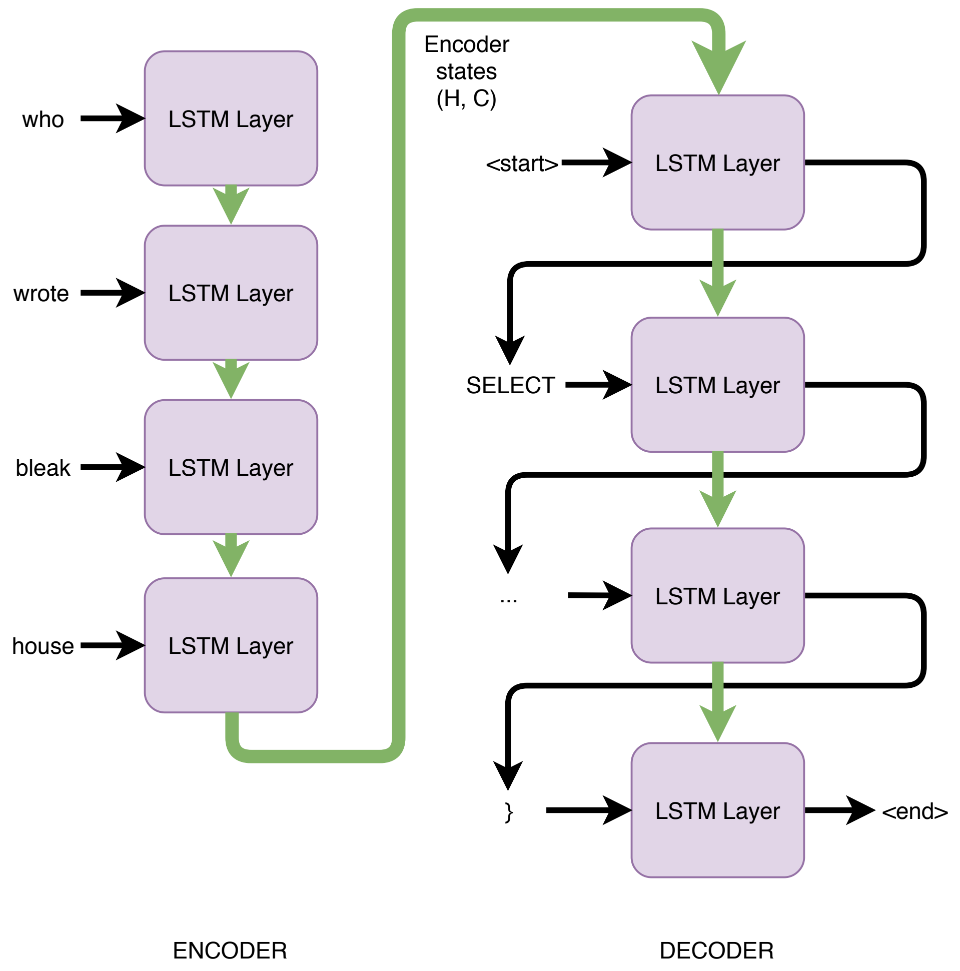

Figure 6: Re-use of the trained model to carry out translation

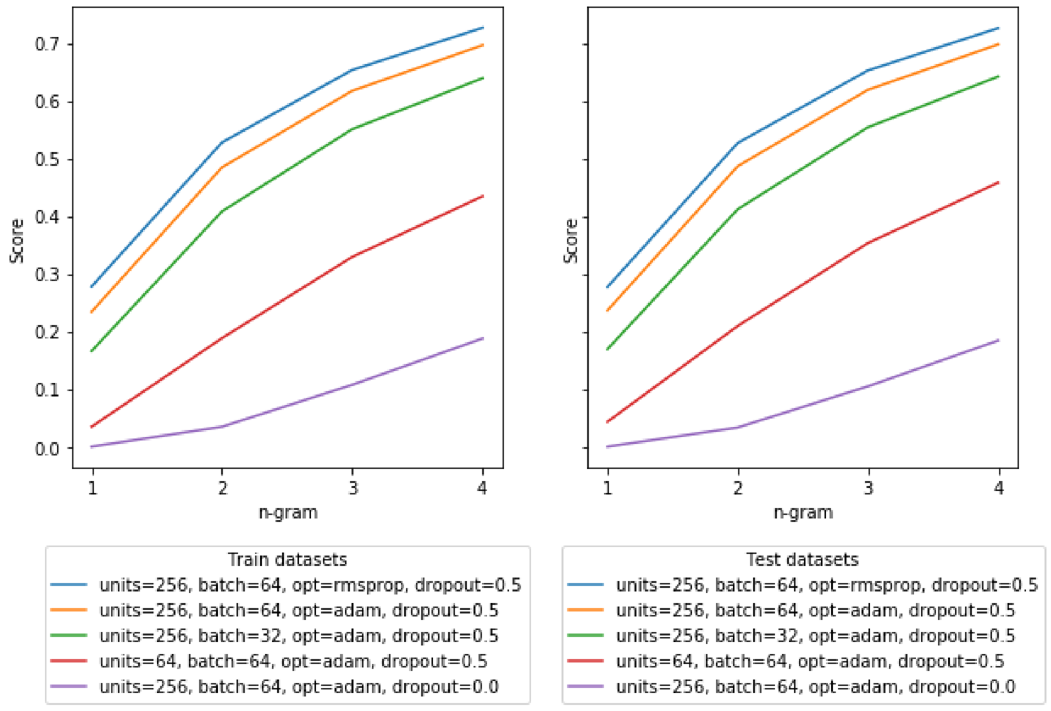

Figure 7: BLEU Metrics

of the model and the small dataset, seems to compare well to the range of 80.89%-99.28% claimed in [7].

Our suspicions should be raised by the similarity between the scores for the training and test sets. Inspection of the translations gives us a clue as to the reason: the model is learning the common words found in the SPARQL queries, and some structural aspects of those queries, but is failing to learn the entities and predicates. The particular subsets of entities and predicates are what vary between any two randomly selected sets; if the model is ignoring them, then it will perform equally well / badly on each of them.

To be of use, the end-to-end approach will need significant refinement in the model, or improvement in the datasets to train on.

5 Conclusion

5.1 Free-Form Visualization

The following are a few sample translations that were produced, produced after the model had been trained with the best hyper-parameter set found:

```
Question:
  what is the team name of st viator high school

Target:
  select distinct ?uri where {
    <http://dbpedia.org/resource/st._viator_high_school> <http://dbpedia.
       org/property/teamname> ?uri
  }

Predicted:
  select distinct ?uri where {
    ?uri <http://dbpedia.org/ontology/associatedband> <http://dbpedia.org/
       resource/foxy_brown_(rapper)>
  }
```

```
Question:
  how many universities are there whose country’s capital is oslo

Target:
  select distinct count(?uri) where {
    ?x <http://dbpedia.org/property/capital> <http://dbpedia.org/resource/
       oslo> .
    ?uri <http://dbpedia.org/ontology/country> ?x .
    ?uri <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia
       .org/ontology/university>
  }

Predicted:
  select distinct count(?uri) where {
    ?uri <http://dbpedia.org/property/team> <http://dbpedia.org/resource/
       chicago_bulls> .
  }
```

You can see that it is predicting valid SPARQL statements, but throwing in totally unrelated entities and predicates. For a given training run, the entities and predicates would be drawn from a small set (usually frequently occurring in the corpus).

Here is an example from when one of the training runs went in the wrong direction:

```
Question:
  name the nearest city to lake victoria

Target:
  select distinct ?uri where {
    <http://dbpedia.org/resource/lake_victoria> <http://dbpedia.org/
       ontology/nearestcity> ?uri
  }

Predicted:
  select distinct ?uri where ?uri . }
```

It is not valid SPARQL, and only consists of keywords, punctuation, or a commonly used variable name ?*uri*.

5.2 Reflection

In summary, these were the project steps:

1. The broad area of interest - searching a knowledge graph - was selected
2. A problem narrow and specific enough in the area was defined (can a fluent search interface be created for an RDF graph)
3. The largest, clean and freely accessible dataset pairing NLQs and SPARQL queries was found
4. The datasets were inspected for their characteristics, and appropriate cleaning and tokenization was selected
5. A survey of research in the area was done, looking for approaches that leveraged something in my machine learning experience e.g. neural networks
6. A published, character-based, non-fixed length translater was adapted to work at the word-level, and constructed with Keras / Tensorflow
7. Different combinations of hyper-parameters were tested, suggesting the optimal combination
8. A SPARQL predicter was used to generate translations, enabling the BLEU metric to examine fluency

Construction of the translater was by far the most challenging aspect, and the time it consumed prevented me for developing the model further. I never found a model implementation of a word-based encoder-decoder that was simple enough as a starting point. Small things like not being aware of the zero_mask parameter cost many days of work!

It was a very useful experience in learning the limitations of neural networks, but it also stimulated many ideas about it can be adapted to solve the problem of search in knowledge graphs.

5.3 Improvement

If you wished to pursue the end-to-end neural machine translation approach, you could consider the following to improve the performance:

1. Increase the dataset size by generating many times more questions with larger entity and predicate sets, and more SPARQL templates
2. Play with deep, LSTM layers (Google Translate reportedly uses networks that are 8 deep) [10]
3. Add features like attention, to improve the "memory" of the network [3]

Translating NLQ to SPARQL has a couple of essential requirements that probably don't apply as much to, say, french to english translations:

1. It must generate valid SPARQL
2. It must return results about the entities in their question

A simple approach to try to guarantee this would be to break the translation down in to stages:

1. Entity extraction, so as to ensure these end up in the SPARQL query
2. Train a model to predict a specific SPARQL template from an NLQ, and populate with the entities (and predicates) extracted

This could be a rabbit hole, the kind that end-to-end NN translation neatly sidesteps, but something only more experimentation will prove.

References

[1] Lisa Ehrlinger and Wolfram Wöß. "Towards a Definition of Knowledge Graphs." In: *SEMANTiCS (Posters, Demos, SuCCESS)*. 2016.

[2] Chen Liang et al. "Neural symbolic machines: Learning semantic parsers on freebase with weak supervision". In: *arXiv preprint arXiv:1611.00020* (2016).

[3] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. "Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025* (2015).

[4] Kishore Papineni et al. "BLEU: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pp. 311–318.

[5] *Question Answering over Linked Data (QALD)*. `http://qald.aksw.org/`.

[6] *Resource Description Framework (RDF)*. `https://www.w3.org/RDF/`.

[7] Tommaso Soru et al. “Neural Machine Translation for Query Construction and Composition”. In: *arXiv preprint arXiv:1806.10478* (2018).

[8] *SPARQL is a recommendation.* https://www.w3.org/blog/SW/2008/01/15/sparql_is_a_recommendation/.

[9] Priyansh Trivedi et al. “Lc-quad: A corpus for complex question answering over knowledge graphs”. In: *International Semantic Web Conference.* Springer. 2017, pp. 210–218.

[10] Yonghui Wu et al. “Google’s neural machine translation system: Bridging the gap between human and machine translation”. In: *arXiv preprint arXiv:1609.08144* (2016).

[11] Mohamed Yahya et al. “Natural Language Questions for the Web of Data”. In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning.* EMNLP-CoNLL ’12. Jeju Island, Korea: Association for Computational Linguistics, 2012, pp. 379–390. URL: http://dl.acm.org/citation.cfm?id=2390948.2390995.