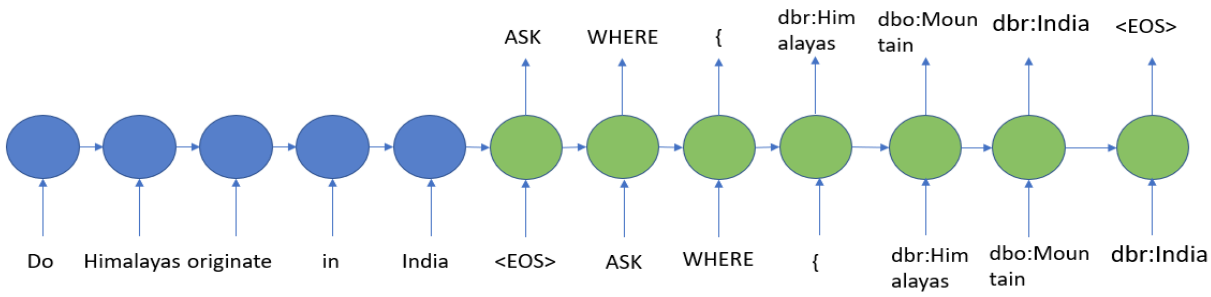
CHALMERS
UNIVERSITY OF TECHNOLOGY



Smart question and answering system using recurrent neural networks

Master of Science in Complex Adaptive Systems

SELVIN CEPHUS JAYAKUMAR

Department of Space, Earth and Environment
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019
Master's Thesis 2019

MASTER'S THESIS 2019

Master of Science in Complex Adaptive Systems

SELVIN CEPHUS JAYAKUMAR

Department of Space, Earth and Environment
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2019

Smart question and answering system using recurrent neural networks
Master of Science in Complex Adaptive Systems

SELVIN CEPHUS JAYAKUMAR

Examensarbete 2019/ Institutionen för Rymd-, geo- och miljövetenskap,
Chalmers tekniska högskola 2019

Department of Space, Earth and Environment
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: + 46 (0)31-772 1000

Cover:
A diagrammatic representation of the RNN used in this these with English input and SPARQL output. For more information see Figure 2.7 in Chapter 2.

Göteborg, Sweden, 2019

Smart question and answering system using recurrent neural networks

Master's thesis in Complex Adaptive Systems

SELVIN CEPHUS JAYAKUMAR
Department of Space, Earth and Environment
Chalmers University of Technology

ABSTRACT

Language translation using RNNs has been around for a long time. The advancement in computer hardware has contributed to its adoption in our day-to-day lives. So far, such translation techniques are used to translate one human language to another. In this thesis we apply these techniques to translating English questions to SPARQL queries. Earlier, similar attempts made focused on achieving accurate translation on datasets consisting of questions from a specific domain. This thesis looks at ways to train an RNN model which is more generic and can learn features of translating an English question to a SPARQL query regardless of the domain. We successfully demonstrate the ability of RNNs to translate questions in English to SPARQL by training the network on tokenised question answer pairs. This results in the reduction of the amount of data required to train such systems as well improves deploying the network in multiple domains after being trained on the tokenized dataset.

Keywords: Natural language processing, Recurrent neural networks, RDFs, SPARQL, Neural machine translation, LSTM, GRU

Contents

Acknowledgement

I would like to thank Semcon AB for giving me this opportunity and support during my time working with them. Special thanks to Peter Nordin, Claes Andersson and the AI team at Semcon for their guidance and supervision. Finally, I would like to thank my family and friends who have supported and encouraged me to keep persevering. Would also like to make a special mention to my first child whom we are expecting in January,2020. Hope he'll read this someday.

Göteborg September 2019

Selvin Cephus Jayakumar

Notations

Acronyms

RNN	Recurrent neural networks
NMT	Neural machine translation
NLP	Natural language processing
RDF	Resource description format
SPARQL	SPARQL Protocol and RDF query language
ANN	Artificial neural network
ML	Machine learning
VA	Virtual assistant
AI	Artificial intelligence
NLU	Natural language understanding
LSTM	Long short-term memory
GRU	Gated recurrent unit

1 BACKGROUND

Communication through language is a unique human tool developed to preserve information over generations. According to Hurford, J.R., 1998, there are different theories as to how language originated, some linguists theorize that language evolved gradually along with human evolution and other theories consider the language faculty to suddenly appear with a chance mutation around 100,000 years ago[1][2]. Regardless of which theory one considers human language took anywhere from a million to a hundred thousand years to develop to its current form. The different languages and dialects can also be attributed to socio-cultural conditions and evolves from one generation to another. One of the key features of language communication is question and answering (QA) and it is just as complex as language itself. It is also the most intuitive way humans get new information and give instructions. For example, 'What is the temperature today? Do I need an umbrella?', 'Can you book an appointment at 2 PM?'. These examples maybe seem familiar as these are some of the most common questions, we ask virtual assistants (VA). VAs are software entities which can perform computer-based tasks by taking natural (human) language as input in the form of text or audio. It uses natural language processing (NLP) which is a subfield of artificial intelligence (AI).

The breakthroughs in artificial intelligence and more specifically artificial neural networks (ANN) has enabled us to teach computers to perform such tasks, albeit simple tasks. ANN algorithms perform a set of calculations in a specific order to get a result based on a specific input. By adjusting the parameters used in the calculations, over a series of time steps one can tune to resulting model to respond with expected answers to questions. Compared to how humans use the ability to ask and answer questions, these approaches are not very flexible. Consider the example about the temperature above, if it was modified to 'I want to go out for a run today, do I need a jacket?'. Here of course, we are not asking for fashion advise, rather we'd like to know the weather conditions so I can make my run as smooth as possible. For a human, this is a simple question but for a computer it is not. One major reason for this is humans have also developed the ability to reason and extrapolate information from things like context. This is also what distinguishes a VA's ability to answer questions from that of a human.

Teaching VAs to reason is also an active field of research today. More and more VAs can use contextual information to answer questions. This can be because of the approach to language learning is moving from an NLP approach, where language is converted to spatial data and the algorithm learns patterns, to a natural language understanding (NLU) approach. In NLU the systems are designed to understand context in which a sentence is used. We try to explore the possibilities of teaching a VA to answer more complex questions using context and reasoning.

The study of natural language in computing started around the 1950s. During this time Alan Turing published a paper called 'Computing Machinery and intelligence' in which he introduces the Turing's test. To summarize, Turing tries to modify the question, 'can a machine think?' to 'is the machine indistinguishable from a human'.

[1] *https://en.wikipedia.org/wiki/Origin_of_language*
[2] *https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4144795/*

In our case, can a machine hear our question, think, and respond correctly and trick the user in believing that the response is coming from a human. This idea that Turing introduces is a good description of what AI today tries to achieve.

We, humans, have developed an ability to understand meaning and context from a question. It is important to note that the meaning and context are not just a direct derivative of the constituent words in a sentence, rather it is also a function of abstract features like human experience and knowledge. Teaching a computer to understand and respond appropriately to a natural language-based question is challenging because of the above factors.

To build an algorithm to understand a sentence requires at the least the following components,

- A lexicon of the language used to create the sentence
- A parser which can break a sentence into smaller phrases based on a set of rules i.e. grammar
- An ontology which describes the relationships between the entities mentioned in the sentence. It also defines the entities.

Building a good ontology is a key ingredient in building such systems and is a laborious task. There are efforts under way to automate the creation of ontologies from chunks of texts. One such tool is FRED [2], which can translate 48 different languages in to linked data or ontologies. Once we can generate ontologies, we can query these using a query language known as SPARQL, which stands for SPARQL Protocol and RDF Query Language. The ontologies are stored in resource description framework (RDF) format.

Expertise in SPARQL is limited to those working with the semantic web. And in order to tap into the rich information stored in the RDFs one needs to be able to write such queries. Luz et al. showed that an LSTM encoder-decoder model with attention can be trained to translate natural language questions into SPARQL. This enables us to build chatbots which can,

a. Take in user queries in natural language,
b. Convert it to SPARQL by passing it through the neural machine translation (NMT) model,
c. Query an RDF database for the correct response,
d. Generate the response in a human readable format and present to the user.

SPARQL-RDF combination enables chatbots to parse more complicated questions. For example, consider the sentence, ‘I want to buy a Tesla, can you help me find some dealers nearby?’. Firstly, the system needs to understand that Tesla in this context is a car, and not a person. And the that I am looking for a car dealer. Since RDFs stores information such as, ‘Tesla is a car.’, ‘People can buy cars.’, ‘Car dealers sell cars’, a SPARQL query can get us the correct information we are looking for in this context.

One of the drawbacks of the models trained in [3] is that they are not generic. They have been trained on a small Geo880 dataset. But it helps us see the potential of using

NMT models that are generally used to translate one language to another in translating a human language to a computer language. In our case, to SPARQL.

1.1 Problem

In our work we explore ways to make these models more generic. One approach is to train an NMT model on templates of English-SPARQL pairs instead of entire sentences. We begin by training a model on a set of English- SPARQL pairs without omitting anything. Unlike in [3] we do not confine these questions to a specific domain. The goal was to see if the model can learn certain salient features. Later, we omit information like specific names and entities in sentence pairs in order to make the dataset more generic in nature and observe the model's performance to correctly generate the SPARQL equivalent of an English question. The results show us that the models indeed learn certain features which distinguishes one type of SPARQL query from another. Generating syntactically correct queries was challenging as our dataset was not large enough to create a generic English-SPARQL NMT model. Hence future work in this direction can yield promising results.

1.2 Related work

In this thesis work the focus has been on exploring ways to make NL-SPARQL translation generic i.e. not domain specific. Earlier work has mostly focused on achieving accurate results on selected datasets from narrowed knowledge areas.

Luz et al. uses LSTM encoder-decoder model with attention to translate NL-SPARQL pairs from a Geo880 dataset. They concluded that the approach yields good results but further tests on a variety of dataset is required.

Yin et al. followed up this work by evaluating eight different NMT models for NL-SPARQL translation. Their work highlighted the need for larger and high-quality datasets for training and CNN based architecture performed the best.

1.3 Research question

"Can we train a recurrent neural network on templates of natural language and SPARQL pairs and create a more generic translation system that is capable of translating NL to SPARQL regardless of the knowledge domain of the question."

2 Recurrent neural networks

RNNs are a family of artificial neural networks which process sequential data i.e., $x^{(1)}, \ldots, x^{(T)}$. It was a result of the work of Rumelhart et al., 1986. Compared to non-recurrent networks like feedforward ANNs, RNNs are capable of processing longer sequences of data. Another feature which makes it useful in tasks like language translation is its ability to process variable length sequences. In language translation the source language and its translation in a target language aren't always of similar lengths and RNNs like LSTMs are widely used in this application.

This section describes some important concepts and theory used in this work.

2.1 Artificial neural networks

ANNs are computing systems that is made up of computational units called neurons. They are inspired from the biological functioning of the animal brain. The neuron in an ANN is a simplified mathematical model of a biological neuron. Just as in biological neurons the dendrites receive input from other neurons and pass it through the axons on to other neurons, in ANNs the input data is fed to the node as input and the node operates on the input based on a predefined function called activation function and if the result is above a certain threshold it activates the neuron.

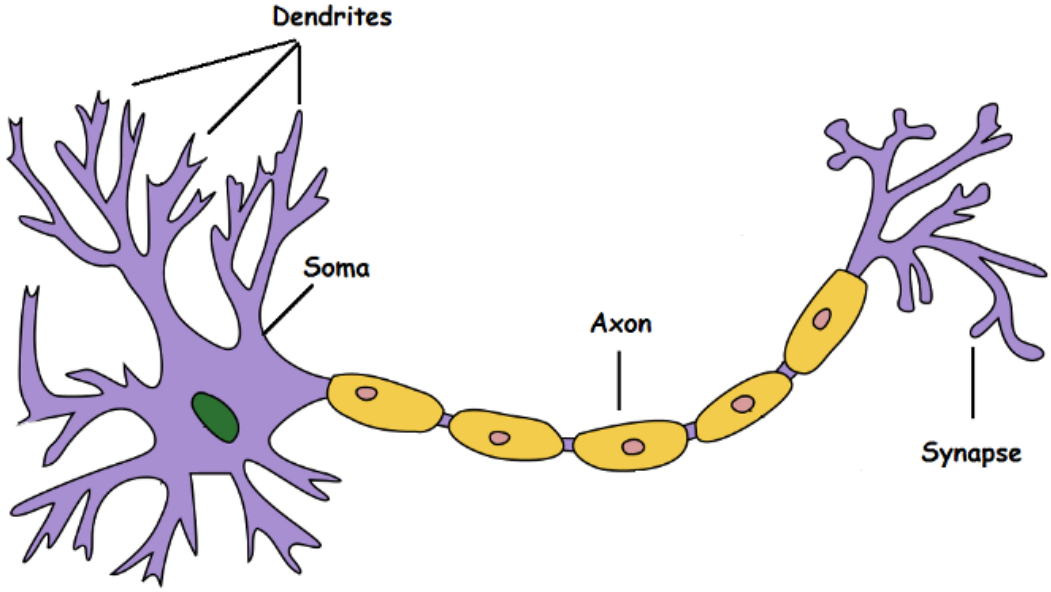


Figure 2.1: Biological neuron[3]

[3] https://commons.wikimedia.org/wiki/File:Neuron_Hand-tuned.svg

A simple model of the artificial neuron was introduced by Warren S. McCulloch and Walter H. Pitts commonly known as McCulloch-Pitts neuron or less commonly as linear threshold gate.

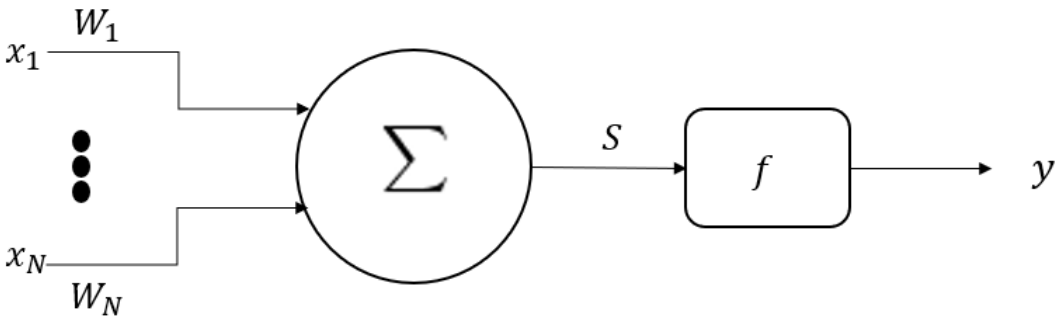


Figure 2.2: McCulloch-Pitts neuron. ***x*** *is the input vector.* ***W*** *denotes the weights that are multiplied with the input.* ***S*** *is the weighted sum of the inputs.* ***f*** *is the activation function and in this case the linear threshold gate;* ***y*** *is the output which is* ***0*** *or* ***1*** *in the case of the McCulloch-Pitts neuron.*

It is known as a linear threshold gate as the ***f*** in Figure 2.2 is a linear threshold function(Figure 2.3) which takes a value of 1 or 0 based on whether ***S*** is above or below a given threshold.

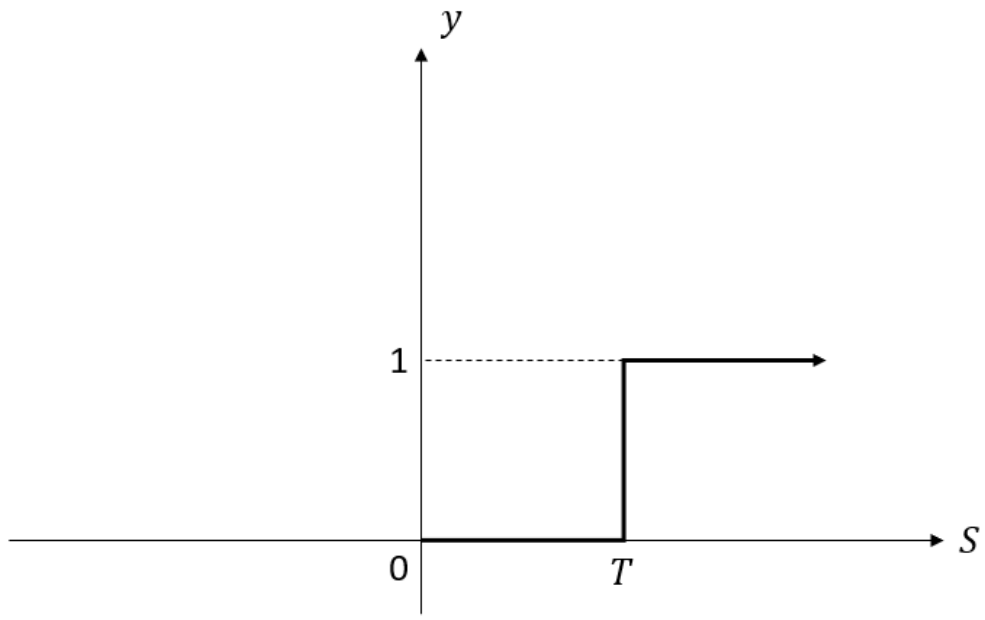


Figure 2.3: Linear threshold function

The McCulloch-Pitts neuron is denoted by the Equation 2.1 and Equation 2.2.

$$\boldsymbol{S} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{W}_i \, \boldsymbol{x}_i \qquad (2.1)$$

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{S}) \qquad (2.2)$$

ANNs like perceptron which is a type of feedforward neural network which is made of neuron units which is a modification of the McCulloch-Pitts neuron. In this case the data is only passed forward from the input to the output and no internal state or memory exists. In contrast RNNs can use their internal state to process a sequence of input where one part of the sequence is dependent on a different part of the sequence.

2.2 Recurrent neural networks

A key difference between a multilayer feedforward network such as a perceptron and a recurrent neural network is that RNNs share parameters across different parts of the model. This makes it possible to generalise RNN models to different lengths.

In applications like natural language processing this is a very useful feature. Consider the sentences, “Sweden celebrates midsummer in June” and “In June, Sweden celebrates midsummer”. When we train a neural network on these examples, we’d like the network to identify June as the month when midsummer is celebrated irrespective of where in the sentence the word June occurs. In contrast a feedforward network would learn separate parameters for each word in the sequence and therefore needs to learn the language grammar at each index of the sentence. RNNs shares the same parameters across several time steps.

2.2.1 Unfolding RNN

Consider the following dynamical system:

$$s^{(t)} = f\left(s^{(t-1)}; \theta\right) \qquad (2.3)$$

where $s^{(t)}$ is called the state of the system.

The system is recurrent since its current state depends on its previous state at $t-1$. We can unfold this system for a finite number of time steps, τ. For example, let $\tau = 3$, we can expand Equation 2.3 recursively as follows,

$$\begin{aligned} s^{(3)} &= f\left(s^{(2)}; \theta\right) \\ &= f(f\left(s^{(1)}; \theta\right); \theta) \end{aligned}$$

Example 2.1: Recursively expanding Equation 2.3 for $\tau = 3$.

The Equation 2.3 can be represented as a directed cyclic computational graph.

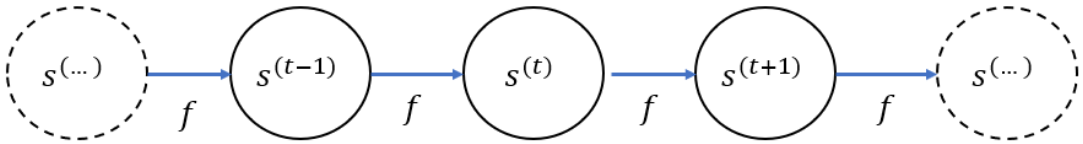


Figure 2.4: Directed cyclic computational graph described in Equation 2.3. $\boldsymbol{s^t}$ is the state of the system at time $\boldsymbol{t}$. The function $\boldsymbol{f}$ maps the consequent states. The parameters of $\boldsymbol{f}$ is denoted by $\boldsymbol{\theta}$ and is the same for all time steps represented in the figure.

Adding an external signal to drive the Equation 2.3 and replacing s with h we get the classical equation that represents the update equation for the hidden state of an RNN.

$$h^{(t)} = f\left(h^{(t-1)}, x^t; \theta\right) \qquad (2.4)$$

Unfolding Equation 2.4 results in the following,

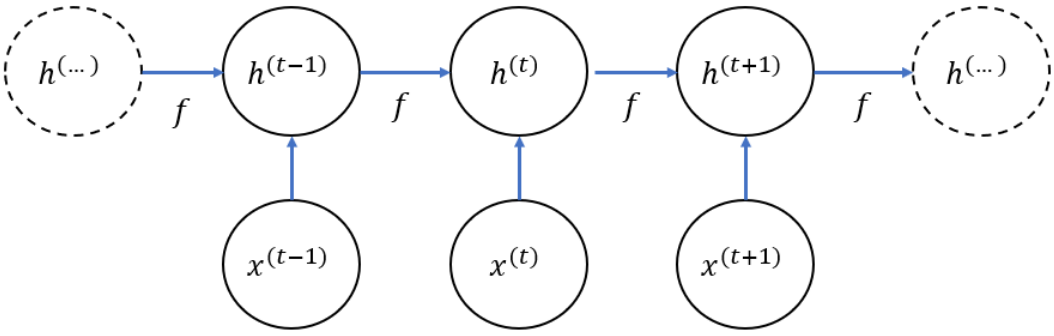


Figure 2.5: Graph representation of a simple recurrent neural network. Where $\boldsymbol{h}$ is the hidden state and $\boldsymbol{x}$ is the input sequence. $\boldsymbol{f}$ maps one hidden state to the next and is parameterised by $\boldsymbol{\theta}$.

2.2.1.1 RNN learning

Training an RNN starts by assigning an initial state, $\boldsymbol{h}^{(0)}$. The forward pass is done by the Equations 2.5 to Equation 2.8.

The input sequence can be used as the initial state or can be connected to the hidden units at each time step for the sequence.

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}_{\boldsymbol{x}}^{(t)} \quad (2.5)$$

$$\boldsymbol{h}^{(t)} = \tanh\left(\boldsymbol{a}^{(t)}\right) \quad (2.6)$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{C} + \boldsymbol{V}\boldsymbol{h}^{(t)} \quad (2.7)$$

$$\hat{\boldsymbol{y}}^{(t)} = softmax\left(\boldsymbol{o}^{(t)}\right) \quad (2.8)$$

$$L\left(\left\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(\tau)}\right\}, \left\{\boldsymbol{y}^{(1)}, \dots, \boldsymbol{y}^{(\tau)}\right\}\right) = \sum_{t} \boldsymbol{L}^{(t)} \quad (2.9)$$

Where, **b, c** are bias vectors
U - weight vector from input to hidden layer
V - Weight vector from hidden layer to output
W - Weight vector from one hidden layer to the next hidden layer
L – Training loss

The Figure 2.5 represents an RNN which processes input and generates output of the same length. The loss is calculated as a sum of losses over all time steps for a given input($\boldsymbol{x}$), output($\boldsymbol{y}$) sequence, represented by Equation 2.9. The gradient of the RNN required for training can be calculated by using the backpropagation through time(BPTT) on the unrolled computational graph using the training loss **L**.

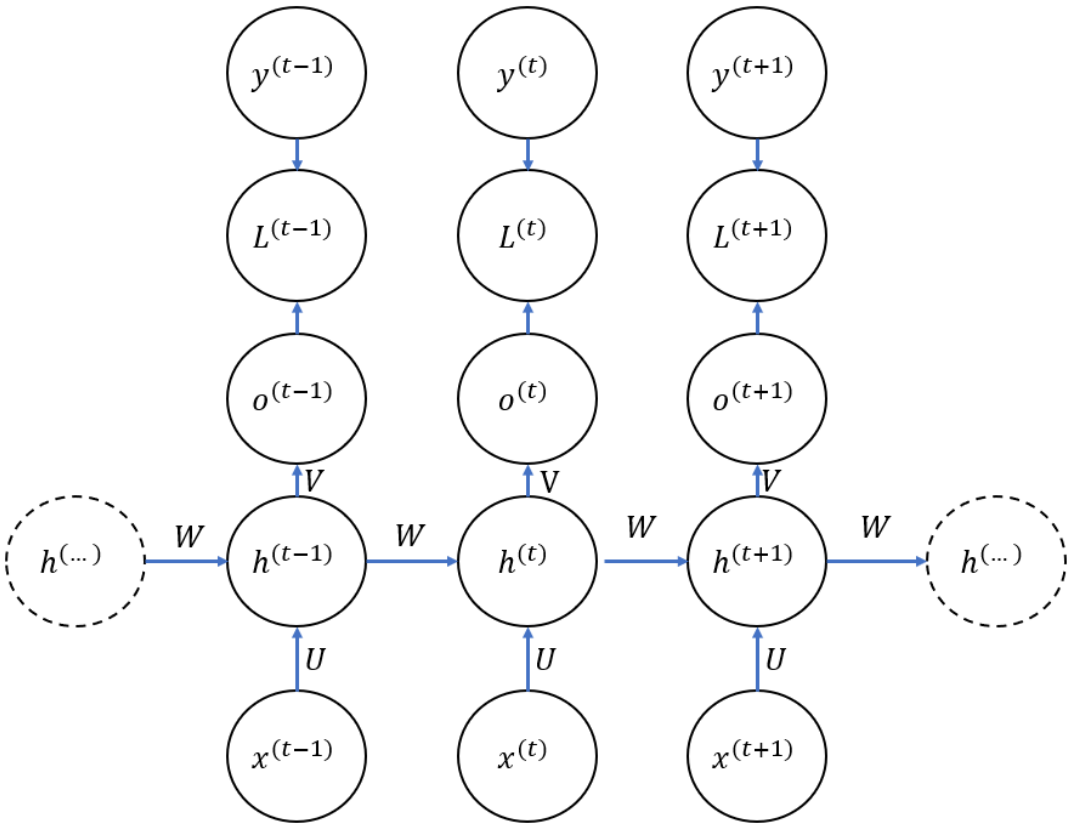


Figure 2.6: The RNN representation which shows how the training loss $\boldsymbol{L}$ *is calculated based on the output* $\boldsymbol{o}$ *of the network and target output* $\boldsymbol{y}$. [4]

2.3 Encoder-decoder sequence to sequence network

RNNs can map an input sequence to a fixed size vector and can also map a fixed size vector to a sequence. This enables us to build encoder-decoder networks or simply seq2seq networks which can be used to map variable size input sequence to another variable size output sequence using the fixed vector as a common reference. Let us explore this with an example from natural language translation. Consider a task to convert sentences from English to Hindi. The encoder processes the English sentence and converts it into a fixed length vector representation. Next, the decoder takes in the vector and converts it in the output sequence in Hindi. By training such a network on English-Hindi sentence pairs we can solve the task using the resulting RNN. Cho et al., Sutskever et al. demonstrated its ability to outperform all other methods used for language translation.

The fixed length vector is called context and is denoted by $\boldsymbol{C}$. Context is considered as the input to the RNN. A limitation of this network architecture is when $\boldsymbol{C}$ has too small a dimension to encode a large input sequence with enough resolution to decode it correctly. Bahdanau et al. proposed a variable length $\boldsymbol{C}$ and also introduced an *attention mechanism* that learns to associate elements of the context vector with the elements of the output sequence.

[4] Deep learning(2017) by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.

Another fundamental problem faced by RNN is the vanishing gradient. When learning long term dependencies gradient tends to either vanish or explode over time as the gradient is carried over many stages or time steps of the RNN. To solve this problem gated RNNs like LSTM and GRU were introduced. We use GRU based RNN in this work.

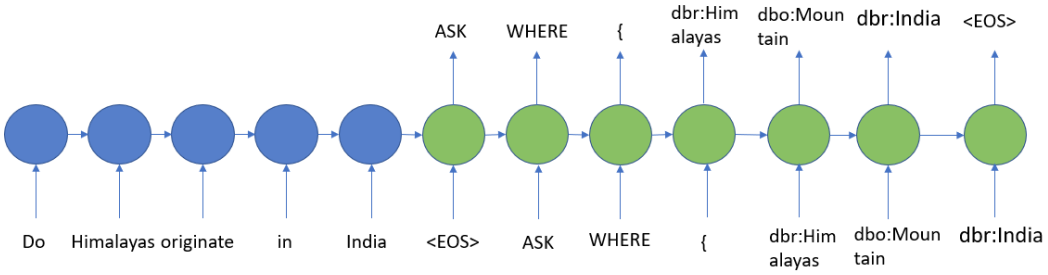


Figure 2.7: An illustration to show how an English to SPARQL translation. The previous input which is at the bottom gives rise to the next input and so on. This thereby generates a single component of the sequence at each node.

2.4 Long-short term memory unit

Long-short term memory networks or LSTMS can successfully handle long term dependencies, thereby solving the vanishing gradient problem with regular RNNs. They were introduced by Hochreiter and Schmidhuber in 1997.

Regular RNNs comprises of a single neural network repeating in a chain like formation. On the other hand, LSTMs have four neural networks that interact with each other in a specific manner. The main output from an LSTM unit is the cell state or context, which is the information passed through the time. The context is updated during each time step by two linear computations. LSTM can decide what information to pass through for each time step using gates. Gates are sigmoid activated neural network layer with a pointwise multiplication operator. This neural network outputs a value of 0 or 1 which in turns controls how much of the gates input is let through. A value of 0 lets nothing through and 1 lets the everything through.

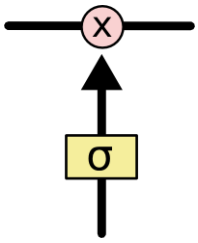


Figure 2.8: An LSTM gate. Illustration is courtesy of Colah's blog on LSTM. [5]

In LSTMs three such gates are used to control how much of the input , $\boldsymbol{x^{(t)}}$, and the previous hidden state, $\boldsymbol{h^{(t-1)}}$, influence the current context $\boldsymbol{C^{(t)}}$. The three gates are denoted by $\boldsymbol{f_t}$ or also known as forget gate, $\boldsymbol{i_t}$ and $\boldsymbol{o_t}$ or simple input and output gates respectively.

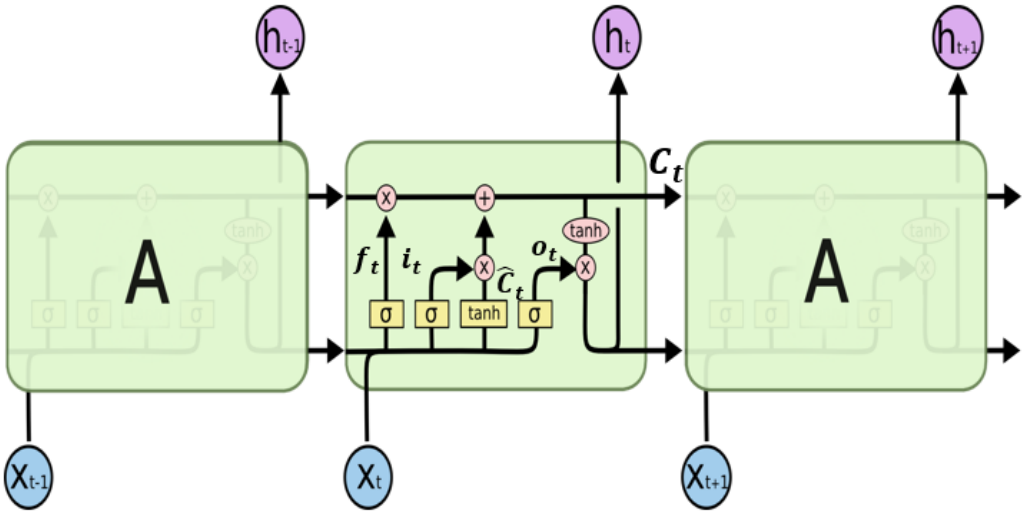


Figure 2.9: LSTM architecture. Illustration is courtesy of Colah's blog on LSTM. [3]

Where,

$$\begin{aligned} \boldsymbol{f^t} &= \boldsymbol{\sigma}\left(\boldsymbol{W^f} \cdot [\boldsymbol{h^{t-1}}, \boldsymbol{x^t}] + \boldsymbol{b^f}\right) \\ \boldsymbol{i^t} &= \boldsymbol{\sigma}(\boldsymbol{W^i} \cdot [\boldsymbol{h^{t-1}}, \boldsymbol{x^t}] + \boldsymbol{b^i}) \\ \boldsymbol{o^t} &= \boldsymbol{\sigma}(\boldsymbol{W^o} \cdot [\boldsymbol{h^{t-1}}, \boldsymbol{x^t}] + \boldsymbol{b^o}) \\ \widehat{\boldsymbol{C}}^{\boldsymbol{t}} &= \tan \mathbf{h}(\boldsymbol{W^C} \cdot [\boldsymbol{h^{t-1}}, \boldsymbol{x^t}] + \boldsymbol{b^C}) \\ \boldsymbol{C^t} &= \boldsymbol{f^t} * \boldsymbol{C^{t-1}} + \boldsymbol{i^t} * \widehat{\boldsymbol{C}}^{\boldsymbol{t}} \end{aligned}$$

W denotes the weights of each gate network.

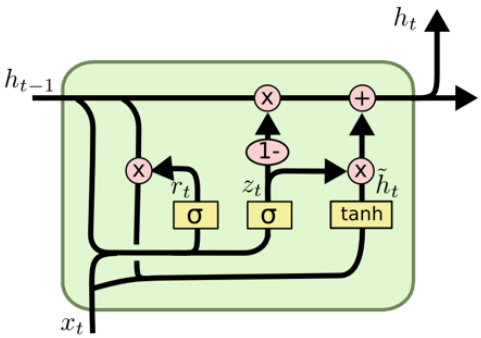
Finally, the current hidden state is calculated using,

$$\boldsymbol{h^t} = \boldsymbol{o^t} * \mathbf{tanh}(\boldsymbol{C^t})$$

[5] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

2.5 Gated recurrent unit

Following the creation of LSTMs other variants have also been introduced. One such variation is the gated recurrent unit or GRU. It was introduced by Cho, et al. in 2014. It is considered a simpler variant as it combined the forget gate and the input gate into a single update gate. Furthermore, the cell state is merged with the hidden state. In this work we use this variant of the LSTM neural network for training.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

*Figure 2.10: GRU architecture. Illustration is courtesy of Colah's blog on LSTM. [3] Where **h** denotes the hidden state, **x** denotes the input, **r** and **z** are output from the gate networks.*

2.6 Encoder

An encoder is a GRU based RNN that takes as input the input sequence, in our case the English language questions and embeds or encodes it into a fixed length vector.

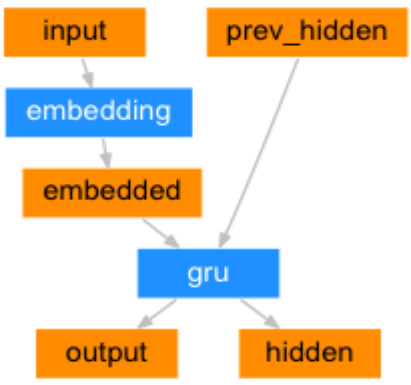


Figure 2.11: GRU based encoder network.[6]

[6] https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

2.7 Decoder

The decoder is also a GRU based RNN that takes the vector, also known as context vector, generated by the encoder, and generates the output sequence in the target language, in our case SPARQL. The decoder is given an input and a hidden state for each time step. The initial hidden state of the decoder RNN is the last hidden state of the encoder, which is the context vector. And the initial input is the start of sentence tag or <SOS>.

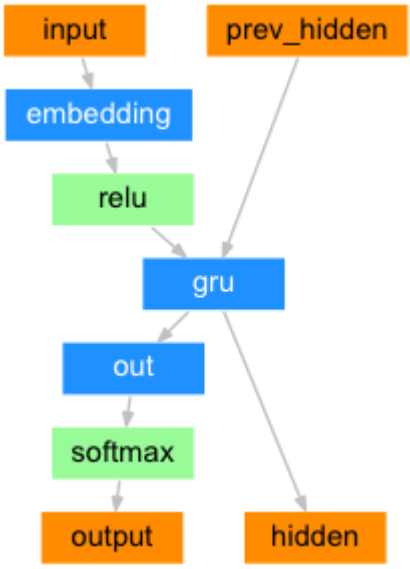


Figure 2.12: GRU based decoder network.[4]

2.8 Attention decoder

The attention decoder is a sophisticated version of the decoder in Figure 2.5. The length of the input sequences can be very large. And if the context vector generated by the encoder does not capture enough information from the input sequence, it might be harder to train longer input-output pairs. Here an attention decoder enables the decoder to focus on other parts of the encoder's output at each time step of the decoder's outputs.

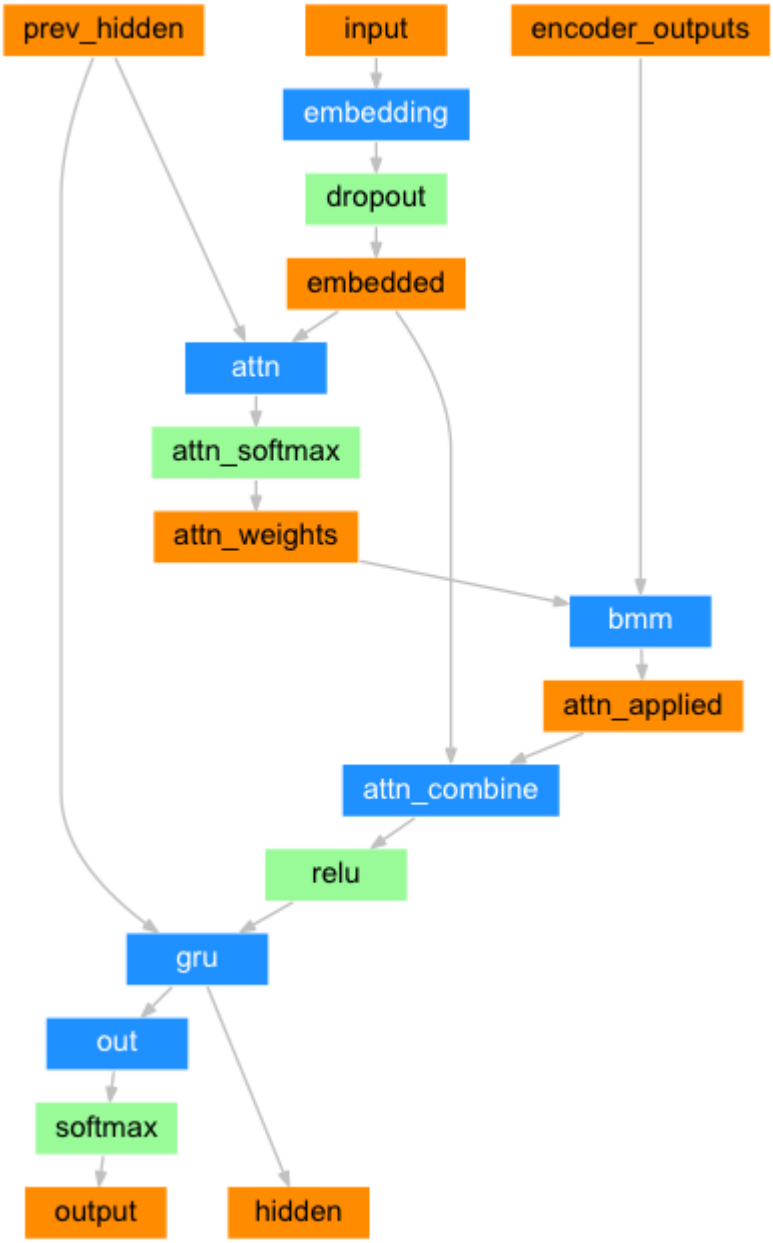


Figure 2.13: Decoder RNN with attention.[4]

3 METHOD

Our goal was to teach a computer writing SPARQL given a question in English. Rather than simply train the RNN models on a large dataset and see how it performed, we set out to observe what features is the model learning. For example, a question which starts with '*How many…*' usually has a SPARQL which has the keyword '*COUNT*'. We wanted to see if our models can learn such fundamental building blocks of SPARQL. We later set out to design an input dataset which omits any unnecessary information to the model which did not contribute to the above objective. The resulting dataset consists of NL-SPARQL pairs where the nouns were replaced by tokens, thereby generating a dataset of templates. Finally, we trained and evaluated the models on this dataset and drew conclusions from the results.

This chapter presents the method used to achieve the above goal. Section 2.1 describes the RNN architecture. In section 2.2 we describe the dataset and the different stages of preparing it. Next, in 2.3 we describe the experimental setup which include the RNN architecture, training and evaluation. Finally, we describe the different software components written to achieve the results.

3.1 Data

This section describes the stages in the gathering and pre-processing of the dataset used in training.

3.1.1 QALD Dataset

The base dataset used in this work is the LC-QuAD v.6[7][8] which is in the Question Answering over Linked Data(QALD) [10] format. QALD is a series of challenges on creating question and answer datasets. The challenge is to convert information available online to the QALD format which enables the use of semantic web query tools like SPARQL to access this information.

A sample of this dataset can be seen in Figure 3.1.

[7] https://figshare.com/articles/LC-QuAD_QALDformat/5818452
[8] http://qald.aksw.org/

3.1.2 Parsing

The training data used for deep learning tasks is in the input-output format. In our case, our inputs are questions in English, and the output is it's SPARQL equivalent.

Our original dataset is not available in this format and consists information we do not need. Hence, we parse the dataset using a python script to extract only the question in English and it's equivalent SPARQL query. The result looks like the example pair in Table 3.1.

Table 3.1 English-SPARQL pair

English question	*Which comic characters are painted by Bill Finger?*
SPARQL	*SELECT DISTINCT ?uri WHERE* *{* *?uri http://dbpedia.org/ontology/creator* *<http://dbpedia.org/resource/Bill_Finger> .* *?uri <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>* *http://dbpedia.org/ontology/ComicsCharacter* *}*
Processed SPARQL	*SELECT DISTINCT ?uri WHERE* *{* *?uri dbo:creator dbr:Bill_Finger .* *?uri rdf:type dbo:ComicsCharacter* *}*

```
{
"dataset": {
"id": "lcquad-v1"
},
"questions": [
{
"hybrid": "false",
"question": [
{
"string": "Which comic characters are painted by Bill Finger?",
"language": "en"
}
],
"onlydbo": true,
"query": {
"sparql": "SELECT DISTINCT ?uri WHERE {?uri <http://dbpedia.org/ontology/creator>
<http://dbpedia.org/resource/Bill_Finger> . ?uri <http://www.w3.org/1999/02/22-rdf-
syntax-ns#type> <http://dbpedia.org/ontology/ComicsCharacter>}"
},
"answers": [
{
"head": {
"vars": [
"uri"
]
},
"results": {
"bindings": [
{
"uri": {
"type": "uri",
"value": "http://dbpedia.org/resource/Batman"
}
},
{
"uri": {
"type": "uri",
"value": "http://dbpedia.org/resource/Alfred_Pennyworth"
}
}
```


Figure 3.1 Sample QALD dataset

3.1.3 Cleaning

Cleaning a dataset for training is perhaps the most important step in data preparation. Providing appropriate information for training determines what the network is likely to learn. We processed the data in a way that emphasizes the format of the SPARQL query. This enables the reconstruction of the query after being generated by the RNN model.

SPARQL queries have links to entities and each entity has a type known as RDF-type. These types can be replaced with shorter keywords which are directly translatable by some SPARQL endpoints. See Table 3.2 for a list of keywords used.

Table 3.2 List of SPARQL tokens

Token	URI
dbo	http://dbpedia.org/ontology/creator
dbr	http://dbpedia.org/resource
rdf:type	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
dbp	http://dbpedia.org/property

3.1.4 Final dataset

The final dataset is a tab separated question-SPARQL pair. Some examples are shown in Table 3.3. This dataset has also been made more generic by replacing the nouns with tokens as in Table 3.4

Table 3.3 Final dataset: Non-tokenized

English	Which comic characters are painted by Bill Finger?

SPARQL	SELECT DISTINCT ?uri WHERE
	{
	?uri dbo:creator dbr:Bill_Finger .
	?uri rdf:type dbo:ComicsCharacter
	}

Table 3.4 Final dataset: Tokenized

English	Which comic characters are painted by **token**?

SPARQL	SELECT DISTINCT ?uri WHERE
	{
	?uri dbo:creator dbr:**token** .
	?uri rdf:type dbo:ComicsCharacter
	}

3.2 Training

This section describes the settings used for training. The actual values of parameters are found in the results section.

3.2.1 Hyperparameters

The following hyperparameters were used in training but only a few of them were modified to achieve better network accuracy.

- Learning rate
- Teacher forcing ratio
- Hidden size
- Number of epochs
- Iterations per epoch
- Total English-SPARQL pairs
- Training-test pairs ratio
- Max sentence length

3.2.2 Cross validation

During each epoch, the order of the dataset was randomized and split into training and test sets at 90% and 10% respectively.

3.2.3 Hardware

At the beginning of the work where the dataset size and epoch run lengths were smaller, a desktop PC equipped with the following specifications was used. CPU: Core i5 3470, 16GB DDR3 RAM, NVIDIA GTX 1070Ti GPU.

During the later stages which had runs which ran for more than 8 hours on the above configuration, I used the free quota available on google cloud platforms compute engine with single GPU configuration. The performance was significantly better.

3.3 Evaluation

The models were evaluated on their ability to generate syntactically correct SPARQL queries for a given question in English. A query was considered syntactically correct if it did not return an error on querying an RDF database endpoint. The Virtuoso SPARQL endpoint[9] was used to evaluate the queries.
Also, queries which we nearly correct and with a minor manual correction can be made error free were also considered as syntactically correct. This was only in cases where a bracket or a dot was missing.

[9] https://dbpedia.org/sparql

3.3.1 Evaluation criteria

The models were trained on the two datasets mentioned in Table 2.3 and Table 2.4. The metric used to evaluate the model was syntactical error.

Accuracy: It is the ratio of queries that are translated correctly in a grammatical sense to the total queries in the test set. Here grammar pertains to the structure of the SPARQL queries. This of course does not necessarily mean that the SPARQL will execute without any error. Hence, we have another metric called syntactical error.
Syntactical error: It is the ratio of queries that a syntactically correct to the total queries in the test set.

$$Syntactical\ error = \frac{Total\ queries - Syntactically\ correct\ queries}{Total\ queries}$$

4 RESULTS

Through the duration of this work each observation has given a clearer picture of what makes a good SPARQL generator. Interestingly the number of samples in the training samples was not the only important factor. Of course, this isn't to say that a large dataset is not important but given that creating a lot of English-SPARQL pairs is a laborious task and finding a way to train an ANN on a smaller dataset and achieve a good deal of generalisation. To avoid writing millions of SPARQL queries for millions of human language questions.

The QALD dataset [9] used was pre-processed in two ways. Generating two different datasets,

- Non-tokenized
- Tokenized

which was used to perform the train and test the RNN. The tests have revealed some interesting observations, which we discuss in detail in this section.

The dataset is split into training and validation sets with a 90% to 10% split during each epoch i.e. during each epoch the dataset is randomly shuffled and split in 90% training pairs and 10% validation pairs.

4.1 Non-tokenized dataset

This dataset contains all 5000 pairs from the final dataset in Chapter 1. This can be considered a brute force approach of training a network to see if the RNN is indeed able to generate accurate sequences of SPARQL queries for a question in English. After some trial and error on the following hyperparameters,

- Hidden size
- Duration of training(iterations/epochs)
- Learning rate

the network managed to achieve really good performance. See Figure 4.1.

Table 4.1: Hyperparameters for best results for non-tokenized dataset

Parameter	Value
Learning rate(average loss > 0.9)	0.01
Learning rate(average loss < 0.9)	0.001
Hidden size(vector size)	1024
Input size(size of English vocab)	8315
Output size(size of SPARQL vocab)	4916
Epochs	10
Iterations/epoch	5000
Total pairs	5000
Training pairs/epoch	4500
Validation pairs/epoch	500
Max length of sentences	30

4.1.1 English to SPARQL sample results for non-tokenized dataset

The following results were generated by our network that was trained on the non-tokenized dataset. As we can observe even when we change the name of the person or place, we are enquiring about the network generates a query based on its closest semantic understanding of the input. Therefore, questions like "Does the west Thurrock come under Essex county?" and "Does Newark come under Essex county?" generate the same SPARQL. This is since the network has not seen examples related to the place called Newark and it only recognizes West Thurrock.

This result motivated us to investigate ways to make the network agnostic of specific information like this and rather train the network to find the closest SPARQL template as we can see in the Section 4.2.

English - Which comic characters are painted by Bill Finger?
SPARQL - SELECT DISTINCT ?uri WHERE
{
?uri dbo:creator dbr:Bill_Finger .
?uri rdf:type dbo:ComicsCharacter
}

English – Was Kevin Jonas a part of Jonas brothers?
SPARQL - ASK WHERE
{
dbr:Jonas_Brothers dbo:formerBandMember dbr:Kevin_Jonas
}

English - Was Nick Jonas a part of Jonas brothers ?
SPARQL - ASK WHERE
{
dbr:Jonas_Brothers dbo:formerBandMember dbr:Kevin_Jonas
}

English - Does the west thurrock come under Essex county ?
SPARQL - ASK WHERE
{
dbr:West_Thurrock dbo:ceremonialCounty dbr:Essex
}

input - Does the Newark come under Essex county ?
SPARQL – ASK WHERE
{
dbr:West_Thurrock dbo:ceremonialCounty dbr:Essex
}

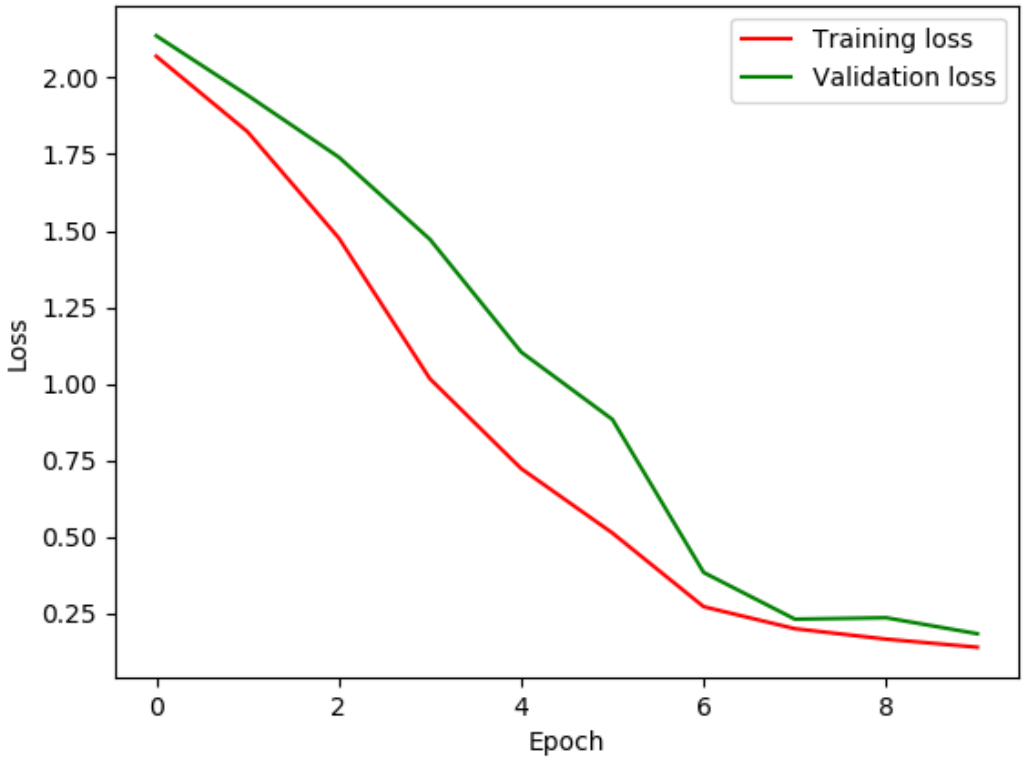


Figure 4.1: Loss graph for hyperparameters in Table 4.2

The model seemed to have associated certain types of question formats to certain SPARQL keywords. For example, questions that started with '*How many…'* included the keyword *COUNT* in the SPARQL output. And questions which started with *'What is…'* included the keyword *WHERE*.

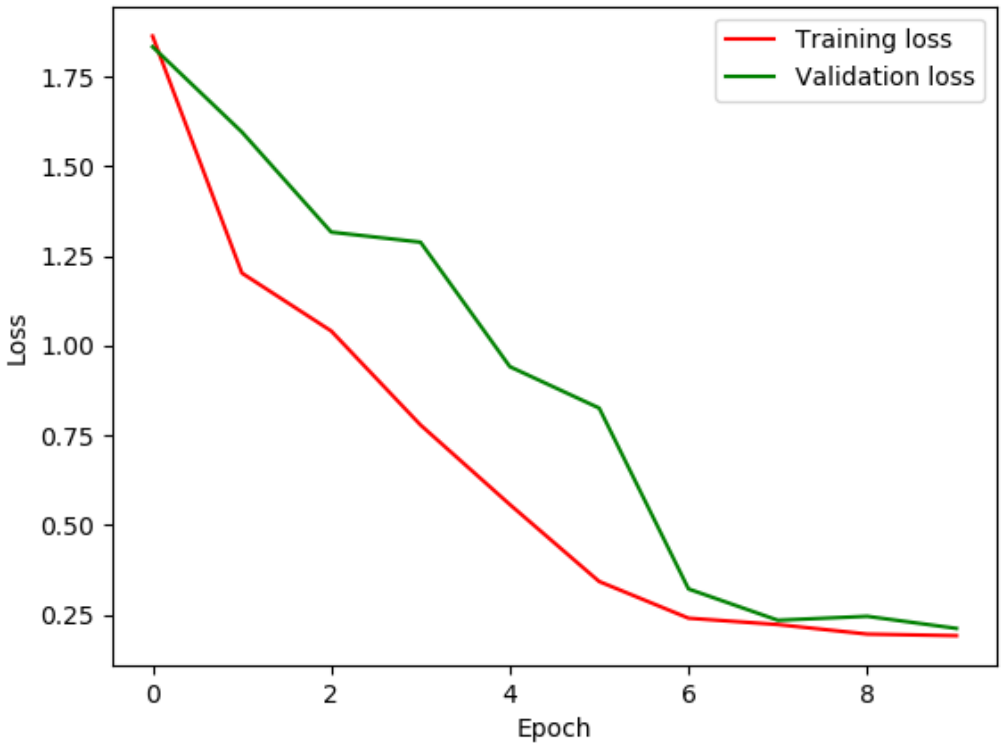
One of the drawbacks of training the RNN model on this dataset was it learned very specific information rather than just SPARQL syntax. Consider the example, *'How many types of RNNs are there?*' . After training the network on this example, we can get an accurate response for a question regarding types of RNNs. But if one wanted to know how many types of cars there are the network with still give a SPARQL response to query types of RNNs. This can be handled by having a postprocessing step to replace '*RNNs'* with '*cars'* but this approach will not scale well.

4.2 Tokenized dataset

In order to make the network generate queries which are generic, we can train the RNNs on a tokenized dataset. Consider the same example as in Section 3.1. *'How many types of RNNs are there?'* , we can replace *RNNs* with a reserved keyword called ***token***. Now our example can be rewritten as, '*How many types of* ***token*** *are there?'*. This will ensure that any question which is like the example would generate a query with the keyword ***token*** in it. And in the postprocessing step we can replace ***token*** with *cars* and generate the desired response.

Table 4.1: Hyperparameters for best results for tokenized dataset

Parameter	Value
Learning rate(average loss > 0.9)	0.01
Learning rate(average loss < 0.9)	0.001
Hidden size(vector size)	256
Input size(size of English vocab)	280
Output size(size of SPARQL vocab)	218
Epochs	10
Iterations/epoch	750
Total pairs	134
Training pairs/epoch	120
Validation pairs/epoch	14
Max length of sentences	15
Syntactical error	0.52



The syntactical error of 0.52 means that out of the 100 test questions the network generated 48 correct SPARQL queries.

4.2.1 English to SPARQL sample results for tokenized dataset

In contrast to the results in Section 4.1.1 here we see that the network generates SPARQL templates for similar questions rather than gives us a query for a wrong person or place as in case with the network trained on non-tokenized dataset.

4.2.1.1 Expected:

English - Was Kevin Jonas a part of Jonas brothers?
SPARQL - ASK WHERE { dbr:Jonas_Brothers dbo:formerBandMember dbr:Kevin_Jonas }

English - Does the west thurrock come under Essex county?
SPARQL - ASK WHERE { dbr:West_Thurrock dbo:ceremonialCounty dbr:Essex }

English - Is Ombla originate in Croatia?
SPARQL - ASK WHERE { dbr:Ombla dbo:sourceMountain dbr:Croatia }

English - Was Ganymede discovered by Galileo Galilei?
SPARQL - ASK WHERE { dbr:Ganymede_(moon) dbp:discoverer dbr:Galileo_Galilei }

English - Was Reza Amrollahi born in Iran?
SPARQL - ASK WHERE { dbr:Reza_Amrollahi dbp:birthplace dbr:Iran }

4.2.1.2 RNN generated:

English - Was Kevin Jonas a part of Jonas brothers ?
SPARQL - ASK WHERE { dbr:Jonas_Brothers dbo:formerBandMember dbr:token}

English - Does the west thurrock come under Essex county ?
SPARQL - ASK WHERE { dbr:token dbo:ceremonialCounty dbr:Essex }

English - Is Ombla originate in Croatia ?
SPARQL - ASK WHERE { dbr:token dbo:sourceMountain dbr:Croatia }

English - Was Ganymede discovered by Galileo Galilei ?
SPARQL - ASK WHERE { dbr:token dbp:discoverer dbr:Galileo_Galilei }

English - Was Reza Amrollahi born in Iran ?
SPARQL - ASK WHERE { dbr:token dbo:assembly dbr:Ethiopia }

5 DISCUSSION

SPARQL along with RDF databases enables us to teach VAs to answer questions that would generally require human reasoning and contextual awareness. This is an alternative to writing all possible question and answer pairs, storing it on a large enough storage and matching a user question to an entry in the database. Compared to the later brute force approach, SPARQL and RDF approach provides a lot more flexibility in asking a question with significantly less information.

But writing SPARQL queries is a laborious task and requires expert knowledge. Hence automating this task would further help developers to train VA on the required task of question answering rather than writing SPARQL queries. One way to automate this task is what was dealt and discussed through this work. As we have seen in numerous cases from the field of language translation, RNNs are good at learning long term dependencies that are required to translate long sequences from one language to another. Similarly, we have applied the principles used in translating one natural language to another, example English to Swedish, using RNNs such as LSTM and GRU to translating English to a computer query language, SPARQL.

We have furthermore explored ways on how to minimize the amount of data required to train such a neural network that generates SPARQL based on an English question. We have seen promising results in generalising the RNN to generating SPARQL while being trained on a smaller dataset. This dataset consisted of tokenised English-SPARQL pairs which was discussed in detail in Section 3.1.

6 Future work

We were able to establish the ability of RNN networks to learn complex patterns in English to SPARQL translation task. Extending the tokenized dataset for training would result in better generalising capability of the network. In our evaluation we tokenized a small portion of the QALD dataset. Extending this to the entire dataset would result in a network capable of covering many more examples of English questions. The tokenization of the dataset was done manually, automating this process would go a long way in achieving bigger and better training dataset.

In our work we limited the tokenization to a single token per training example. Introducing advanced tokenization methods could help standardise this process and enable further research in the area of tokenization. To improve the performance of the RNN, we also recommend experimenting with various cross-validation methods in order to achieve lower training error.

7 References

Hurford, J.R., 1998. The evolution of language and languages. In *The evolution of culture*. Edinburgh University Press.

Gangemi, A., Presutti, V., Reforgiato Recupero, D., Nuzzolese, A.G., Draicchio, F. and Mongiovì, M., 2017. Semantic web machine reading with FRED. *Semantic Web*, *8*(6), pp.873-893.

Luz, F.F. and Finger, M., 2018. Semantic parsing natural language into sparql: improving target language representation with neural attention. *arXiv preprint arXiv:1803.04329*.

Turing, A.M., 2009. Computing machinery and intelligence. In *Parsing the Turing Test* (pp. 23-65). Springer, Dordrecht.

Yin, X., Gromann, D. and Rudolph, S., 2019. Neural Machine Translating from Natural Language to SPARQL. *arXiv preprint arXiv:1906.09302*.

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*.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. Nature, 323, 533-536.

Sutskever, I., Vinyals, O. and Le, Q.V., 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (pp. 3104-3112).

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

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