Deep Neural Semantic Parsing: Translating from Natural Language into SPARQL

Fabiano Ferreira Luz

Thesis Presented
to the
Institute of Mathematics and Statistics
of the
University of São Paulo
in partial fulfillment
of the
requirements for the Degree of
Doctor of Science

Program: Graduate in Computer Science

Advisor: Prof. Dr. Marcelo Finger

Financial Support from CNPq

São Paulo, October 2018

Deep Neural Semantic Parsing: Translating from Natural Language into SPARQL

This is the original version of the thesis written by
the candidate Fabiano Ferreira Luz
submitted to the Judging Committee.

Acknowledgements

I would like to thank my advisor Marcelo Finger for allowing me to explore new approaches to semantic parsing, for his support and constructive criticism of my work. I would like to thank all my friends and family who have been on my side during the development of this work. There are so many friends and loved ones that I will not even try to name just to not commit the rudeness of forgetting someone.

Abstract

Semantic parsing is the process of mapping a natural-language sentence into a machine-readable, formal representation of its meaning. The LSTM Encoder-Decoder is a neural architecture with the ability to map a source language into a target one. We are interested in the problem of mapping natural language into SPARQL queries, and we seek to contribute with strategies that do not rely on handcrafted rules, high-quality lexicons, manually-built templates or other handmade complex structures. In this context, we present two contributions to the problem of semantic parsing departing from the LSTM encoder-decoder. While natural language has well defined vector representation methods that use a very large volume of texts, formal languages, like SPARQL queries, suffer from lack of suitable methods for vector representation. In the first contribution we improve the representation of SPARQL vectors. We start by obtaining an alignment matrix between the two vocabularies, natural language and SPARQL terms, which allows us to refine a vectorial representation of SPARQL items. With this refinement we obtained better results in the posterior training for the semantic parsing model. In the second contribution we propose a neural architecture, that we call Encoder CFG-Decoder, whose output conforms to a given context-free grammar. Unlike the traditional LSTM encoder-decoder, our model provides a grammatical guarantee for the mapping process, which is particularly important for practical cases where grammatical errors can cause critical failures. Results confirm that any output generated by our model obeys the given CFG, and we observe a translation accuracy improvement when compared with other results from the literature.

Keywords: NLP, RNN, LSTM, SPARQL, RDF, CFG, Word Embeddings, Encoder Decoder, Semantic Parsing, Ontology, Grammars.

Resumo

A análise semântica é o processo de mapear uma sentença em linguagem natural para uma representação formal, interpretável por máquina, do seu significado. O LSTM Encoder-Decoder é uma arquitetura de rede neural com a capacidade de mapear uma sequência de origem para uma sequência de destino. Estamos interessados no problema de mapear a linguagem natural em consultas SPARQL e procuramos contribuir com estratégias que não dependam de regras artesanais, léxico de alta qualidade, modelos construídos manualmente ou outras estruturas complexas feitas à mão. Neste contexto, apresentamos duas contribuições para o problema de análise semântica partindo da arquitetura LSTM Encoder-Decoder. Enquanto para a linguagem natural existem métodos de representação vetorial bem definidos que usam um volume muito grande de textos, as linguagens formais, como as consultas SPARQL, sofrem com a falta de métodos adequados para representação vetorial. Na primeira contribuição, melhoramos a representação dos vetores SPARQL. Começamos obtendo uma matriz de alinhamento entre os dois vocabulários, linguagem natural e termos SPARQL, o que nos permite refinar uma representação vetorial dos termos SPARQL. Com esse refinamento, obtivemos melhores resultados no treinamento posterior para o modelo de análise semântica. Na segunda contribuição, propomos uma arquitetura neural, que chamamos de Encoder CFG-Decoder, cuja saída está de acordo com uma determinada gramática livre de contexto. Ao contrário do modelo tradicional LSTM Encoder-Decoder, nosso modelo fornece uma garantia gramatical para o processo de mapeamento, o que é particularmente importante para casos práticos nos quais erros gramaticais podem causar falhas críticas em um compilador ou interpretador. Os resultados confirmam que qualquer resultado gerado pelo nosso modelo obedece à CFG dada, e observamos uma melhora na precisão da tradução quando comparada com outros resultados da literatura.

Keywords: PLN, RNN, LSTM, SPARQL, RDF, GLC, Palavras Associadas, Codificação Decodificação, Análise Semântica, Ontologias, Gramáticas.

Contents

Abbreviations list

NN	*Neural Network*
NLP	*Natural Language Processing*
OWL	*Ontology Web Language*
RDF	*Resource Description Framework*
CEC	*Constant Error Carrousel*
MRL	*Meaning Representation Language*
WER	*Word Error Rate*
CFG	*Context Free Grammar*
DCG	*Definite Clause Grammar*
RNN	*Recurrent Neural Network*
ANN	*Artificial Neural Network*
LSTM	*Long Short-Term Memory*
SPARQL	*SPARQL Protocol and RDF Query Language*
POSTAG	*Part Of Speach Tag*
BiRNN	*Bidirectional Recurrent Neural Network*

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and goals

One of the main challenges of natural language processing (NLP) has always been the development of techniques that allow natural interaction between humans and machines. Communication between humans and machines is being improved, in so far as human language is better understood and processed by those machines. This task is very difficult and is divided into subtasks (Winograd, 1972). Extracting the meaning (or meanings) from sentences is an essential part of this process. Among the fields studied by computational linguistics, semantic parsing is the one directly related to this task.

Semantic parsing can be defined as the process of mapping natural language sentences into a machine-interpreted, formal representation of its meaning. The idea of constructing a formal representation to a sentence in natural language is old, Montague grammar (Bach, 1979, Rodman, 1972) was considered a landmark of its time (Bach, 1979). Richard Montague has developed a whole theory where it makes a relation between the semantics of natural language and its grammar, that is is based on formal logic, especially higher-order predicate logic and lambda calculus. One of the most important features of the model developed by Montague is the principle of semantic compositionality, that is, the meaning of the whole is a function of the meanings of its parts and their mode of syntactic combination. Richard Montague's work inspired a number of other successful approaches.

Traditional approaches to semantic parsing, based on sophisticated grammars, rely on high-quality lexicons, manually-built templates, and linguistic features which are either domain or representation-specific (Dong and Lapata, 2016). Due to the expensive construction and maintenance of traditional models, the approaches based on supervised Machine Learning have become more attractive. To the extent that these approaches depend on little or no linguistic knowledge, they become appropriate to the current scenario where there exists a large volume of data available (Ge and Mooney, 2005) and knowledge of specialists is much more expensive.

The objective of this project is to contribute to the evolution of models of semantic parsing that do not rely on handcrafted rules, high-quality lexicons, manually-built templates or other handmade complex structures. The neural network approach using RNN Encoder-Decoder with LSTM has been successful in several NLP tasks (Cho et al., 2014a, Graves, 2012, Sutskever et al., 2014). More specifically, such architecture has considerably advanced the state-of-the-art in semantic parsing (Dong and Lapata, 2016), thus motivating its use in this work.

In view of our general objective, the starting point for our research is the LSTM Encoder-Decoder model (Cho et al., 2014b) similar to what was used in the (Bahdanau et al., 2014, Dong and Lapata, 2016). From then on, we evolved two aspects of this model, in the first contribution, we focus on improving the vectorial representation of the target language lexicon, in our case, the SPARQL query language. The choice of SPARQL language is due to the practicality and popularization of language as a fundamental element in the so-called Semantic Web (Arenas and Pérez, 2011, Bizer and Schultz, 2009, Buil-Aranda et al., 2013, Calvanese et al., 2017, Hartig et al., 2009, Hitzler et al.,

2009). The second contribution of our work is in the developing and implementation of a version of the model called Encoder-CFG Decoder which guarantees that the output of the decoder is in agreement with a given context free grammar.

Mapping sentences to a logical form is a central problem in designing natural language interfaces (Zettlemoyer and Collins, 2005). We discuss the experimental results on two database domains: Geo880, a set of 880 queries to a database of United States geography; and Jobs640, a set of 640 queries to a database of job listings. The works Tang and Mooney (2001), Zettlemoyer and Collins (2005) described results on these data sets. Works by Thompson (2003), Zelle and Mooney (1996) used a subset of the Geo880 corpus. Regarding the transformation of natural language queries in SPARQL we compared our work with AlAgha (2015).

We evaluated our model accuracy in returning entirely correct logical forms for each test sentence.

1.2 Contribution

The task of transforming natural language queries to query language is not a trivial task (Androutsopoulos et al., 1995, Capetta, 2012, Tablan et al., 2008, Warren and Pereira, 1982) . Our approach is based on a vector representation for the inputs of a recurrent neural network (Encoder Decoder). Thus, the main contributions are the following:

- A vector representation suitable for a formal language, in our case SPARQL;
- The creation of ontologies and a SPARQL version for the datasets: geo880 and job640;
- The modeling, coding, training and evaluation of a neural network capable of learning to encode questions in natural language and decode SPARQL query;
- The development of the concept of Encoder-CFG Decoder used to ensure the syntacticity in the output sentences of the network.

1.3 Thesis Structure

In Chapter 2 we have a brief literature review of the concepts used in the development of this work, already in the Chapter 3 let's talk about the main work related. In Chapter 4 we explain our proposed approach. In Chapter 5 we discuss preliminary tests and in the Chapter 6 we make our considerations about all the work and we also talk about future works.

Chapter 2

Background

This chapter is devoted to a literature review of the concepts used in our work. We present a brief description of the techniques and technologies used and at the end of each description we highlight the importance of the concept in the context of our work.

2.1 Semantic Parsing

Semantic parsing is defined by (Mooney, 2007) as the mapping of a natural language (NL) sentence into a complete, formal meaning representation (MR) or logical form. A meaning representation language (MRL) is a formal unambiguous language that allows for automated inference and processing, such as first-order predicate logic.

Semantic Parsing is the central problem in many tasks, such as question answering, robot control, ontology learning, etc .The most usual approach used in semantic parsing are based on categorial grammars. Currently, there are many efforts to build models of semantic parsing fully based on machine learning (Dong and Lapata, 2016). Several tasks can involve the concept of semantic parsing:

- **Robot control**: RoboCup[1] is an international AI research initiative using robotic soccer as its primary domain. In the Coach Competition, teams of agents compete on a simulated soccer field and receive advice from a team coach in a formal language called CLang. The robots in the simulator can interpret the CLang instructions which then affect their behavior while playing the game. The semantic parsers that have been developed for this MRL were part of a larger research project on advice-taking reinforcement learners that can accept advice stated in natural language. Other applications involving robots have been developed (Lauria et al., 2002, Matuszek et al., 2013).
- Ontology learning: Some ontology learning approaches aim at the extraction of the meaning from a certain text and then save this data. The processes of extraction can be adapted into a semantic parsing problem (Biemann, 2005).
- Question answering systems: There are several initiatives of softwares that promise to give an answer to questions posed in natural language. Many of them are available on the Internet (Cao et al., 2011, Ferré, 2014, Katz, 1997).
- Intelligent agents: software agents sometimes need to read and interpret text to perform certain tasks (Chang et al., 2015).

2.1.1 Machine Learning and Semantic Parsing

The works involving Machine Learning and Semantic Parsing is moving towards in at least two different directions. On the one hand, works that develop hybrid logical machine learning

[1] www.robocup.org

approaches (Zettlemoyer and Collins, 2005) . On the other hand, works that try to learn the whole process without use of lexicons or rules handwritten (Dong and Lapata, 2016). In this section, we will explore methods that respect the second approach. Works with this second approach usually make use of probabilistic methods, then, learning a logical form given a natural language expression generally follows the equation:

$$p(a|q) = \prod_{t=1}^{|a|} p(y_t|y_1, ..., y_{t-1}, q), \tag{2.1}$$

where $q = (w_1, ..., w_{|q|})$ is a natural language sequence and $a = (y_1, ..., y_{|a|})$ is the target language sequence. So, here we will deal with the problem of learning a sequence given another input sequence. Deep architectures that mainly use LSTM neural networks are fairly efficient at learning sequences and structures (Malhotra et al., 2015, Tai et al., 2015).

Since approaches that depend on handwritten rules are usually very hard to perform and tied to a specific domain, techniques of machine learning are very welcome for solving natural language processing problems. In this context, researchers seek to develop a solution as automated as possible. Regarding semantic parsing as a statistical translation problem has delivered some interesting results(Dong and Lapata, 2016).

A promising approach for automating statistical translation processes is the deployment of neural machine translation. To better understand the task of semantic parsing being treated with an extension of the neural machine translation model one needs to understand some of the mechanisms used in this concept. The following sections are devoted to the description of some concepts involved in the art of neural machine translation.

2.2 Recurrent Neural Networks

Artificial Neural Networks (ANNs) has become an important concept in the field of machine learning in recent years. Basic information about ANNs can be found in the Appendix A.

A Recurrent Neural Network (RNN) is a type of artificial neural network in which the connections between the units form a directed cycle. This cycle creates an internal state network that allows networks simulate dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use its internal memory to process arbitrary sequences of inputs. This task becomes applicable to the task that involves sequence learning, such as in handwriting recognition, translation and in voice recognition.

On the simple feedforward architecture neural network the outputs depended only on the current inputs. On the recurrent neural networks the outputs depend on a set of previous used inputs[2].

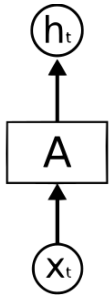


Figure 2.1: *Feedforward NN flow*

[2]Every the pictures of this section as well the simplify explanations were extracted of the web address : http://colah.github.io/posts/2015-08-Understanding-LSTMs/

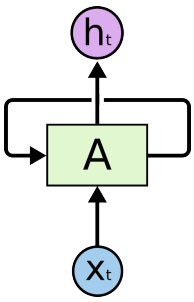


Figure 2.2: *Recurrent NN flow*

The recurrent neural networks are an important tool used in learning sequences, but in practice they are difficult to be trained (Hochreiter et al., 2001). This difficulty is due to the fact that the applicants networks are prone to vanishing gradient problem.

Vanishing gradient: This problem is a difficulty in the training of neural network with methods based in descending gradient and backpropagation. In such methods, each of the neural network weights receives an update proportional to the gradient of the error function with respect to the current weight of each training iteration. Traditional activation functions, such as the hyperbolic tangent function have gradients in the range (-1, 1) or [0, 1) and in the backpropagation the gradients are calculated by the chain rule. This has the effect of multiplication of n small numbers to calculate the gradients of the layers "front" in an n-layer network, which means that the gradient decreases exponentially with n and the front layers are trained very slowly.

With the advent of backpropagation algorithm in the 1970s, many researchers tried to train deep neural network supervised from scratch, initially with little success. Then, the work developed by (Hochreiter, 1991, Hochreiter et al., 2001) have been identified formally the reason for this failure "vanishing gradient problem", which not only affects networks feedforward many levels, but also the recurrent neural networks. In the case of recurrent networks the problem occurs because the recursive nature of this architecture. The training problem of recurrent networks are addressed by a the Long Short-Term Memory (LSTM) neural network architecture.

Long Short-Term Memory

The Long Short Term Memory networks (LSTM) are a special kind of RNN, capable of learning long-term dependencies. They were introduced by (Hochreiter and Schmidhuber, 1997), and were refined and popularized by many people in following works. They work very well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

Step by step: Like the traditional RNNs, the LSTMs networks also have recurrence, but the repeater module has a very different structure. Instead of a single-layer neural network, LSTMs have four, which interact in a very special way.

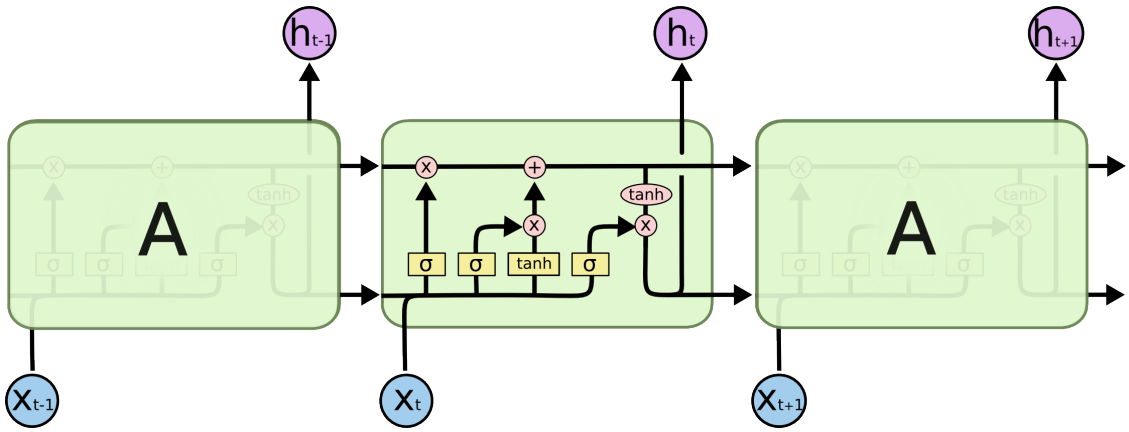


Figure 2.3: *LSTM Gates*

In a Figure 2.3, each line represents an entire vector, from the output of one node to the inputs of the others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denotes that its content is being copied and that the copies are going to different locations.

The key to the operation of LSTM is the construction of a unit called "memory cell". The memory cell may be seen in Figure 2.4. It is a horizontal line that cuts across the top of the diagram. Each memory cell is built aroud a central linear unit with a fixed self-connection (Hochreiter and Schmidhuber, 1997). This architecture allows information to pass through the memory cell without being modified by irrelevant information.

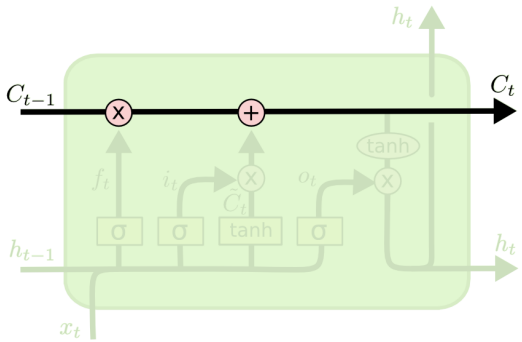
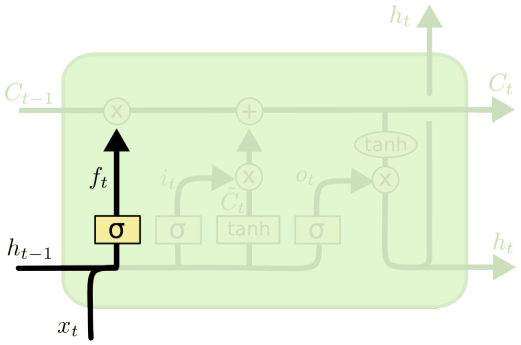


Figure 2.4: *The repeating module in an LSTM contains four interacting layers*

The LSTMs have the ability to remove or add information to the memory cell, this process is carefully regulated by structures called *gates*. Gates are tools to allow optionally information through. They are composed of a neural network with sigmoid activation function and the multiplication operator. The sigmoid function has an image that allow only values between zero and one. We can interpret these values as follows: when the output value of this unit is zero, it means " not pass anything"; when the output value is equal to 1 so it means " let go completely ". An LSTM has three of these gates, to protect and control the memory cell.

The first step in LSTM is to decide how much information we will lose in the memory cell. This decision is made by a sigmoid layer called the *forget gate layer* f_t. It receives as input h_{t-1}, x_t, and b_f (bias), and outputs a number between 0 and 1 for each number in the memory cell C_{t-1}. The output 1 represents "completely keep this" while the output 0 represents "completely get rid of this."

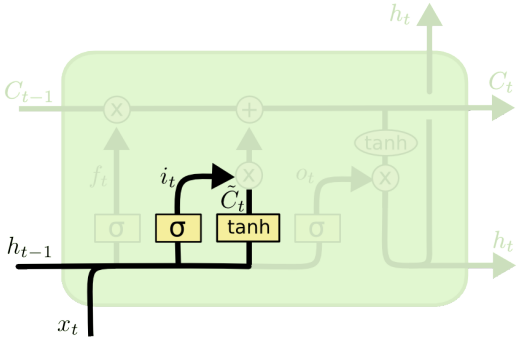
Let's go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want it to forget the gender of the old subject.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 2.5: *Forget gate layer*

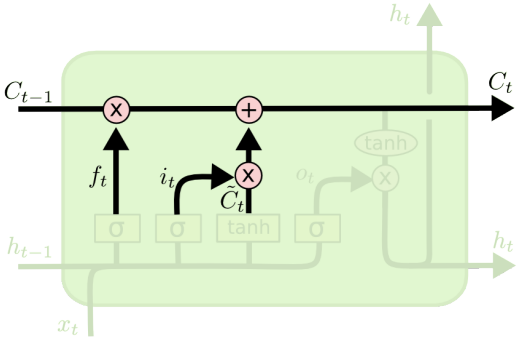
The next step of LSTM is to decide how much of the new information will be included in the memory cell. This has two parts. First, a sigmoid layer called the *input gate layer* i_t decides which values will be updated. Next, a *tanh* layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, these two are combined to create an update to the state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 2.6: *Input gate layer*

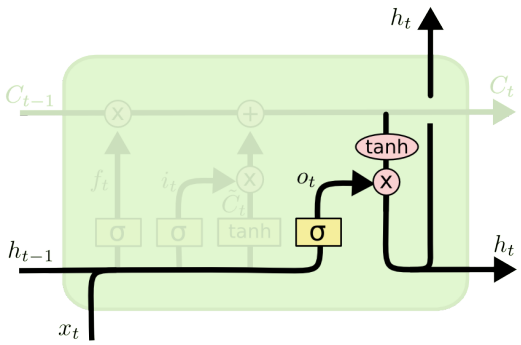
The previous steps are sufficient to change the old state C_{t-1}, generating the new state C_t. We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. Those are the new candidate values to C_t, scaled by how much information we decided to update each state value.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.7: *Memory cell unit*

Finally, we need to generate the output value of h_t. This output is based on the value of the already updated memory cell. First, we run a sigmoid layer called *output gate* o_t which decides what parts of the memory cell will be going to the output. Then, we put the memory cell through tanh(getting values between -1 and 1) and multiply it by the output of the *output gate*, so that we only output the parts we decided to.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2.8: *Output gate layer*

Why LSTM works? In short, in Figure 2.7, we can observe that the memory cell is the main component of this model. Therefore the memory is made up from the information we store or lose, depending on the inputs and the trains we made with them. The sigma functions are controllers, acting as "taps"; its weights and inputs learn how much information must be forgotten, how much will be recorded in the memory and how much goes to the hidden layer.

How does the network learn what it shall forget, what it shall include and how it shall calculate the new hidden layer? The main idea can be described as follows: the supervised learning is a clear example of a general input that has its own expected output; therefore the network will adjust the weights to deliver the desired output. The network can either lose information quickly or maintain the old information for an extended period of time; this feature helps with the weights' adjustment, because the network considers the words and disregard the recent ones. Hence, the network itself determines which words will be important during the learning process. You can understand it clearly by simply looking at the formulas widespread throughout the current section.

Bidirectional Recurrent Neural Network

The Bidirectional architecture of neural netwoks was develop by (Schuster and Paliwal, 1997), using a finite sequence to predict or label each element of the sequence based on both the past and the future contexts of the element. This is done by concatenating the outputs of two RNN, one processing the sequence from left to right, the other one from right to left. The combined outputs are the predictions of the teacher-given target signals. This technique proved to be especially useful when combined with LSTM RNN.

The bi-directional model has been demonstrating success in some applications (e.g. in speech recognition (Graves et al., 2013), as well as on translation (Bahdanau et al., 2014)).

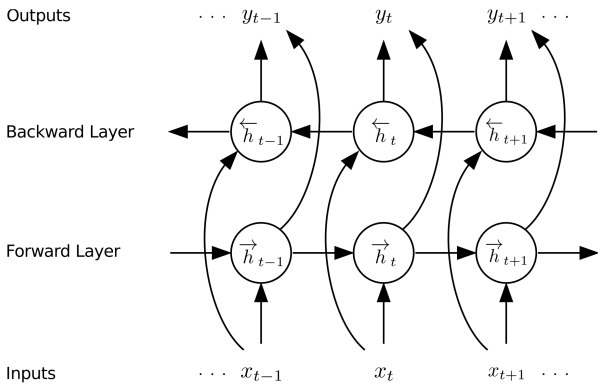


Figure 2.9: *Bidirectional Neural Network (Graves et al., 2013)*

A bidirectional RNN consists of two RNN, the first network have feed forward and the second is

backward. The forward RNN $\overrightarrow{f}$ reads the input sequence as it is ordered from x_1 to x_{T_x} and calculates a sequence of *forward hidden states* $(\overrightarrow{h}_1, ..., \overrightarrow{h}_{T_x})$. The backward RNN $\overleftarrow{f}$ reads the sequence in reverse order, from x_{T_x} to x_1, resulting in a sequence of *backward hidden states* $(\overleftarrow{h}_1, ..., \overleftarrow{h}_{T_x})$. The bidirectional architecture is important in the construction of attention mechanisms applied to artificial neural networks.

Softmax Function

The softmax function is very useful in the context of our work, is also known as exponential normalized (Bishop, 2006). This function is a generalization of the logistic function that "squashes" a K-dimensional vector $\mathbf{z}$ of arbitrary real values to a K-dimensional vector $\sigma(\mathbf{z})$ of real values in the range (0, 1) that add up to 1. The function is given by:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1, ..., K.$$

The softmax function is used in various probabilistic multiclass classification methods including multinomial logistic regression, multiclass linear discriminant analysis, naive Bayes classifiers and artificial neural networks. Specifically, in multinomial logistic regression and linear discriminant analysis, the input to the function is the result of K distinct linear functions, and the predicted probability for the j'th class given a sample vector $\mathbf{x}$ is:

$$p(y = j|x) = \frac{e^{\mathbf{x}^T \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^T \mathbf{w}_k}} \tag{2.2}$$

This can be seen as the composition of K linear functions $\mathbf{x} \mapsto \mathbf{x}^T\mathbf{w}_1, \ldots, \mathbf{x} \mapsto \mathbf{x}^T\mathbf{w}_K$ and the softmax function (where $\mathbf{x}^T\mathbf{w}$ denotes the inner product of $\mathbf{x}$ and $\mathbf{w}$).

To better understand this functions in all of its details, we better depart from a simple method, the logistic regression, which is totally related with the obtainment of the softmax function. when we isolate the linear equation from the logistic function we obtain the following equation:

$$\ln\left(\frac{p(x)}{1 - p(x)}\right) = \theta^T \mathbf{x} \tag{2.3}$$

The set $\theta^T\mathbf{x}$ can be expanded and written as the function $f(k, i) = \theta_{0,k} + \theta_{1,k}x_{1,i} + \theta_{2,k}x_{2,i} + ... + \theta_{M,k}x_{M,i}$, where $\theta_{m,k}$ is a regression coefficient associated with the m-th explanatory variable and with the k-th outcome class. As explained in the logistic regression, the regression coefficients and explanatory variables are normally grouped into vectors of size $M + 1$, so that the predictor function can be written more compactly. Regarding the logistic regression the value k equals 2; in the equation the probability $p(x)$ can be associated with the probability $P(Y_i = 1)$ and the term $(1 - p(x))$ associated to $P(Y_i = 0)$. When we extend it to a broadest set of categories (say, k with $k > 2$) we obtain:

$$\ln \frac{p(Y_i = 1)}{p(Y_i = K)} = \theta_1^T \mathbf{x}_i$$
$$\ln \frac{p(Y_i = 2)}{p(Y_i = K)} = \theta_2^T \mathbf{x}_i$$
$$....$$
$$\ln \frac{p(Y_i = K - 1)}{p(Y_i = K)} = \theta_{K-1}^T \mathbf{x}_i$$

As we showed above, we introduced a separate set of regression coefficients, each one relating to a specific outcome. Now we will isolate the probabilities, beggining by the application of the exponential function to both sides of the equation and then put the term $p(Y_i = K)$ on the other

side of the equation.

$$p(Y_i = 1) = p(Y_i = K)e^{\theta_1^T \mathbf{x}_i}$$
$$p(Y_i = 2) = p(Y_i = K)e^{\theta_2^T \mathbf{x}_i}$$
$$....$$
$$p(Y_i = K-1) = p(Y_i = K)e^{\theta_{K-1}^T \mathbf{x}_i}$$

Now we will sum all the terms to obtain the probability of K. We can also observe that the sum of all the probabilities must necessarily yield 1.

$$p(Y_i = K) = \frac{1}{1 + \sum_{k=1}^{K-1} e^{\theta_k^T \mathbf{x}_i}}$$

Now we can use this general equation to find probabilities to specific cases:

$$p(Y_i = 1) = \frac{e^{\theta_1^T \mathbf{x}_i}}{1 + \sum_{k=1}^{K-1} e^{\theta_k^T \mathbf{x}_i}}$$

The softmax function estimates the probability of Y_i be of a certain category K given a vector $\mathbf{x}_i$ that represents the features of and a weight vector that must be estimated. It is typically jointly estimated by a maximum a posteriori (MAP) estimate, which is an extension of the maximum likelihood using the regularization of the weights to prevent bad solutions [3]. The solution can be found by the deployment of an iterative procedure, such as the descending gradient algorithm (because the function is differentiable).

In neural network simulations, the softmax function is often implemented at the final layer of a network used for classification. Such networks are then trained under a log loss (or cross-entropy) regime, giving a non-linear variant of multinomial logistic regression.

2.3 ANNs and Computational Linguistics

Currently, neural networks have become increasingly popular for the task of language modeling. But, the idea of using neural networks to describe language is old, the first significant attempt was made by (Elman, 1990). The first relevant attempt to build a statistical neural network based language model of real natural language, along with an empirical comparison of performance to standard techniques (n-gram models and class based models) was probably done by Yoshua Bengio in (Bengio et al., 2003). Holger Schwenk, who showed that NNLMs work very well in a state of the art speech recognition systems, and are complementary to standard n-gram models (Tomáš, 2012).

In the first model, Bengio uses feed-forward neural networks. Whereas feed-forward networks only exploit a fixed context length to predict the next word of a sequence, conceptually, standard recurrent neural networks can take into account all of the predecessor words. The contributions of the work (Tomáš, 2012) are remarkable, it uses the recurrent neural networks and makes a generalization on the amount of input parameters among others improvements. In the following section, we present a summary of the Bengio model, an important model that will help us to understand the obtainment of word vector representation in a distributed space.

2.3.1 Word representation in Vector Spaces

The idea if representing the meaning of words in a geometric space dates back to a long time (Suci and Tannenbaum, 1957). In that paper the authors use the technique of classifying subjects in a series of scales whose ends were represented by opposite poles (for example, slow-fast). These ratings were also processed with a dimensionality reduction technique to discover the latent semantic

[3] Usually a squared regularizing function, which is equivalent to placing a zero-mean Gaussian prior distribution on the weights; other distributions are also possible.

structure. Unfortunately, this approach required the classification of various subjects to create a representation to each word, which in practice limited the semantic space for a small number of words.

Another important milestone in the evolution of this technique are the more recent semantic space models, such as Latent Semantic Analysis (Deerwester et al., 1990) and Hyperspace Analogue to Language (Lund and Burgess, 1996). These models intended to overcome the limitation from the previous models by constructing semantic representations indirectly from real language corpora. Its was based on work (Suci and Tannenbaum, 1957) and the well-know vector space model used in information retrieval (Salton et al., 1975). Several other proposed and evaluated models can be found on the literature. Despite their differences, they are all based on the same premise: Words occurring within similar contexts are semantically similar (Harris, 1968).

The advantage of taking such a geometric approach is that the similarity of word meanings can be easily quantified by measuring their distance in the vector space, usualy cosine of the angle between them (Mitchell and Lapata, 2010) (Equação 2.4). In addition, the vector space representation has other advantages such as allowing working with real values, rankings and apply other algebraic properties.

$$sim(A, B) = \cos(\theta) = \frac{A.B}{\|A\|\|B\|} \tag{2.4}$$

The representation of words by vectors in a semantic space distributed has been very promising in recent years. The works (Mikolov et al., 2013a,b,c) have shown interesting linguistic regularities that can be captured by these vectors. For example, by using a word offset technique where simple algebraic operations are performed on the word vectors, it was shown that vector("King") - vector("Man") + vector("Woman") results in a vector that is closest to the vector representation of the word "Queen".

Other interesting relations of the cited works are: Finding a word that is similar to "small" in the same sense as "biggest" is similar to "big", they simply compute vector X = vector("biggest")-vector("big") + vector("small"). Then, they search in the vector space for the word closest to X measured by cosine distance, and use it as the answer to the question (they discard the input question words during this search). When the word vectors are well trained, it is possible to find the correct answer (word "smallest") by using this method.

Figure 2.10 illustrates the model's ability to automatically organize concepts and implicitly learn the relationships between them. During the training, it is not provided any information on the concepts of city and capital. Figure 2.10 was plotted in two dimensions using PCA of the 1000-dimensional Skip-gram vectors(Guthrie et al., 2006) of countries and their capital cities.

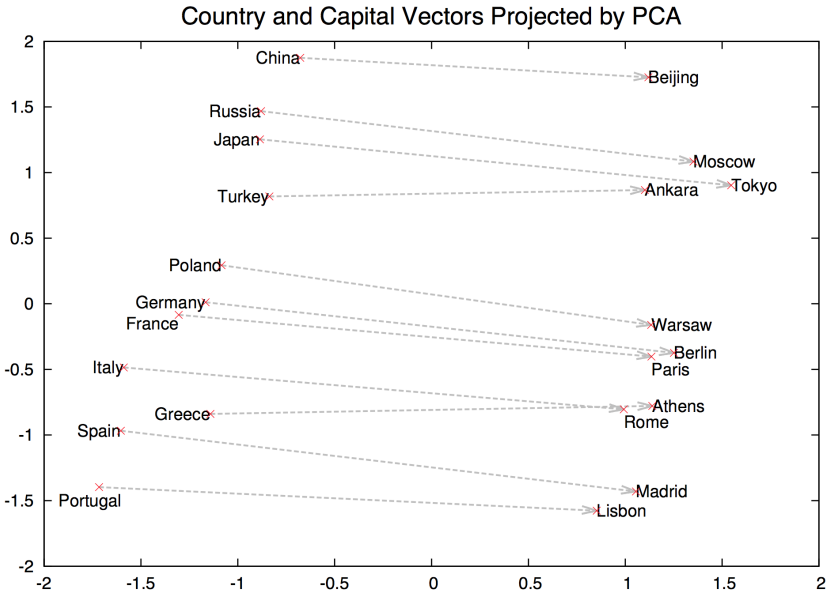


Figure 2.10: *Two-dimensional PCA projection (Mikolov et al., 2013b).*

An inherent limitation of word representations is their indifference to word order and their inability to represent idiomatic phrases. For example, the meanings of "Canada" and "Air" cannot be easily combined to obtain "Air Canada" (Mikolov et al., 2013b). There are some approaches to mitigate this problem (Mikolov et al., 2013b, Mitchell and Lapata, 2010), in general, they are related to the employment of more efficient compositionality types. For example, when we add vectors, the order of the factors does not influence the result. However, some compositions may change the value of representation, such as the tensor product of two vectors. Often, the use of compositional representation methods can be interesting. In the next section we will talk about compositional representation distributed.

2.3.2 Encoder Decoder to Translate Problem

The fundamental feature of a RNN is that it contains at least one feed-back connection, so the activations can flow round in a loop. That enables the networks to do temporal processing and learn sequences, e.g. performing sequence recognition/reproduction or temporal association/prediction.

For simple architectures and deterministic activation function, learning can be obtained by using gradient descent procedures similar to those used in back-propagation algorithm to the feed-forward neural networks. When activations are stochastic, simulated annealing approaches can fit better.

A RNN is a neural network composed by a hidden state h and an optional output y that is the result of a variable-length sequence $x = (x_1, ..., x_T)$. At each time step t, the hidden state h_t of the RNN is updated by equation:

$$h_t = f(h_{t-1}, x_t), \tag{2.5}$$

where f is a non-linear activation function. This function may be as simple logistic sigmoid function and/or a complex function as a long short-term memory (LSTM) unit (Hochreiter and Schmidhuber, 1997). A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop (see Figure 2.11).

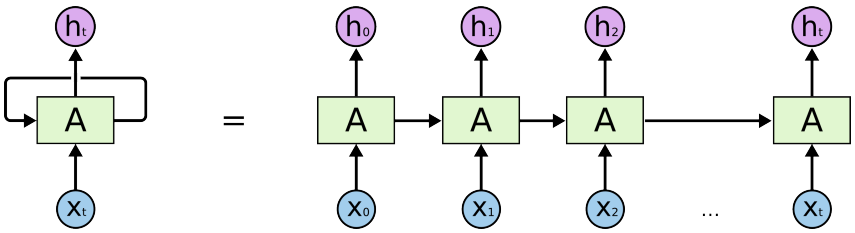


Figure 2.11: *An unrolled recurrent neural network*

A RNN can learn a probability distribution over a sequence by training to predict the next symbol in a sequence. In such case, the output at each time step t is the conditional distribution $p(x_t|x_{t-1}, ..., x_1)$. For example, a multinomial distribution (1-of-K coding) can be obtained through the use of a softmax activation function:

$$p(x_{t,j} = 1|x_{t-1}, ..., x_1) = \frac{\exp(w_j h_t)}{\sum_{j'=1}^{K} \exp(w_{j'} h_t)}, \tag{2.6}$$

for all possible symbols $j = 1, ..., K$, where w_j are the rows of a weight matrix W. By combining these probabilities, we can compute the probability of the sequence x using the equation below:

$$p(x) = \prod_{t=1}^{T} p(x_t|x_{t-1}, ..., x_1). \tag{2.7}$$

From this learned distribution, it is easy to sample a new sequence by iteratively sampling a symbol at each time step (Cho et al., 2014b).

Encoder-Decoder: Cho et al. (2014b) proposed a novel neural network architecture that learns to *encode* a variable-length sequence into a fixed-length vector representation and to *decode* a given fixed-length vector representation back into a variable-length sequence.

From a probabilistic point of view, this new model is a general method to learn the conditional distribution of a conditioning variable-length sequence in another variable-length sequence. This distribution may be interpreted as $p(y_1, ..., y_S|x_1, ..., x_T)$, we can note that the input and output sequence length T and S can be different.

The *encoder* is a RNN that reads each symbol of an input sequence x sequentially. As it reads each symbol, the hidden state of the RNN is updated according to equation (2.5). After reading the end of the sequence (marked with an end sequence symbol), the hidden state of the RNN is summarized all the input sequence. We can call this summary of c. In order to simplify we can define c as the output y.

The *decoder* of the model proposed in (Cho et al., 2014b) is another RNN which is trained to generate the output sequence by predicting the next symbol y_t given the hidden state h_t. However, unlike the RNN described previously, here both y_t and h_t are conditioned to y_{t-1} and summary c of the input sequence. Thus, the hidden state of the decoder at time t is computed by:

$$h_t = f(h_{t-1}, y_{t-1}, c),$$

and likewise, we can define the conditional distribution of the next symbol by the following equation:

$$P(y_t|y_{t-1}, ..., y_1, c) = g(h_t, y_{t-1}, c). \tag{2.8}$$

The g activation function should produce valid probabilities, in this case can be used the softmax function for example. See Figure 2.12 a graph showing the overview of the encoder and decoder scheme.

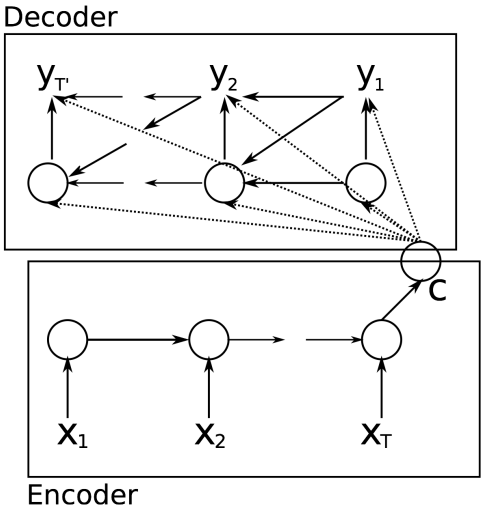


Figure 2.12: *Encoder-Decoder scheme (Cho et al., 2014b).*

The combination of the two described components (Encoder and Decoder) make up the training of the proposed model to maximize the conditional log-likelihood and it can be represented by the equation:

$$\max_{\theta} \frac{1}{N} \sum_{n=1}^{N} \log p_\theta(y_n|x_n), \tag{2.9}$$

where θ is the set of the model parameter and each pair (x_n, y_n) is respectively an input sequence and an output. In our case, we use the vector representation of questions in natural language as input and the SPARQL query as an output. As the output of the decoder, starting from the input, is differentiable, we can use a gradient-based algorithm to estimate the model parameters.

After training the encoder-decoder RNN, the model can be used in two distinct ways. In the first case, we can use the model to generate a target sequence, once the input sequence is provided. In the second the model can be used to evaluate a given pair of input and output sequences by calculating a kind of score (for example, where the score may simply be a probability $p_\theta(y|x)$).

Soft Alignment with Neural Attention

The concept of "attention" has gained popularity recently in training neural networks, enabling the models to learn alignments between different problems. For example, between speech frames and text in the speech recognitions (Chorowski et al., 2014), between image objects and agent actions in the dynamic controls (Mnih et al., 2014), or between visual features of a picture and its text description in image caption generations (Xu et al., 2015).

In the context of machine translations using neural network, (Bahdanau et al., 2014) have successfully applied such attentional mechanism to jointly translate and align words. Here, we are particularly interested here in the latter modality. In the work (Bahdanau et al., 2014), the authors speculate that the encoder-decoder model using a summarized vector of fixed size is a bottleneck for performance improvement. In (Cho et al., 2014a) the authors showed that indeed the performance of a basic encoder-decoder deteriorates rapidly as the length of an input sentence increases.

Given these problems (Bahdanau et al., 2014) propose an extension of the encoder-decoder model which learns alignment and translation at the same time. Each time the model generates a word in translation, the model does a search (soft) for a set of positions in a source sentence where the relevant information is concentrated then the model predicts the target word.

The model proposed by (Bahdanau et al., 2014) differs from the basic model of encoder-decoder by not attempting to encode a full entry into a fixed-size vector. Instead of doing so it encodes entries into a sequence of vectors and selects a subset of them adaptively while encoding the translation.

Through this improvement, the responsibility of compressing information of an entire entry in a fixed-size vector no longer lies at the neural network. The new architecture proposed by (Bahdanau et al., 2014) consits of a bidirectional RNN as an encoder and a decoder that emulates searching through a source sentence while decoding a translation.

New Decoder: In a new model architecture, each conditional probability is defined by:

$$p(y_i|y_1, ..., y_{i-1}, x) = g(y_{i-1}, s_i, c_i), \tag{2.10}$$

where s_i is an RNN hidden state for time i, computed by:

$$s_i = f(s_{i-1,y_{i-1},c_i})$$

we may note that unlike the basic encoder-decoder approach (see equation (2.8)), here the probability is conditioned on a distinct context vector c_i for each target word y_i.

The vector context c_i depends on a sequence of *annotations* $(h_1, ..., h_{T_x})$ in which an encoder maps the input sentence. Each annotation h_i consist of information about the whole input sequence with a strong focus on the parts surrounding the i-th word of the input sequence.

The vector context c_i, is, then computed as a weighted sum of these annotations h_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{2.11}$$

The weight α_{ij} of each annotation h_j is computed by:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}, \tag{2.12}$$

where

$$e_{ij} = a(s_{i-1}, h_j)$$

is an alignment model that scores how well the inputs around position j and the output at position i match. The score is based on the RNN hidden state s_{i-1} (just before emitting y_i, equation (2.10)) and the j-th annotation h_j of the input sentence.

The alignment model a is parametrized as a feedforward neural network which is concomitantly trained with all the other components of the proposed system. The alignment model directly computes a soft alignment, which allows the gradient of the cost function to be backpropagated through. This gradient can be used to train the alignment model as well as the whole translation model at the same time.

The use of a weighted sum of all annotations can be interpreted as calculating an *expected annotations*, where the expectation is over possible alignments. Let α_{ij} be a probability that the target word y_i is aligned to, or translated from, a source word x_j. Then, the i-th context vector c_i is the expected annotation over all the annotations with probabilities α_{ij}.

The probability α_{ij}, or its associated energy e_{ij}, reflects the importance of the annotation h_j to the previous hidden state s_{i-1} for deciding the next state s_i and generating y_i this implements a mechanism of attention in the decoder. The decoder decides parts of the source sentence to pay attention to. By using the attention mechanism in the decoder component, the encoder is relieved because now it does not need to encode all input information in a fixed-size array. By this new approach the information can be spread throughout the sequence of annotations, which can be selectively retrieved by the decoder accordingly.

New Encoder: The proposed encoder in (Bahdanau et al., 2014) doesn't uses a standard RNN described in equation (2.5). The standard edition reads a input sequence x starting from the first element x_1 to the last x_{T_x}. However, in the proposed scheme, the annotation h_j each word does not use only the summary of the previous word but also the following words. Then, an appropriate

neural network is bidirectional RNN (Schuster and Paliwal, 1997), in which has been used recently to speech recognition (Graves et al., 2013) succeeding.

A bidirectional RNN described in section 2.2 forward RNN $\overrightarrow{f}$ part that reads the input sequence as it is ordered from x_1 to x_{T_x} and calculates a sequence of *forward hidden states* $(\overrightarrow{h}_1, ..., \overrightarrow{h}_{T_x})$ and the backward RNN $\overleftarrow{f}$ part which reads the sequence in reverse order, from x_{T_x} to x_1, resulting in a sequence of *backward hidden states* $(\overleftarrow{h}_1, ..., \overleftarrow{h}_{T_x})$.

The annotation is obtained for each word x_j by concatenating the forward hidden state $\overrightarrow{h_j}$ and the backward one $\overleftarrow{h_j}$, for example:

$$h_j = \left[\overrightarrow{h}_j^T; \overleftarrow{h}_j^T\right]^T \tag{2.13}$$

Thus, the annotation h_j of both words (precedings and followings). Due to the tendency of RNNs to better represent recent inputs, the annotation h_j will be focused on the words around x_j. This sequence of annotations is used by the decoder and the alignment model later to compute the context vector.

Mathematical Intuition The basic idea of the attention mechanism for neural networks is inspired by the human visual attention, where the goal oriented (top dow) and stimulus driven (bottom up) attention mechanisms determine the region of focus in the visual input. Among the hypotheses that support the development of a dual brain attention mechanism are: limit the amount of processing a portion of the visual field of cognitive interest (top-down) and also to assist attention as important (salient) at any point in visual field (Corbetta and Shulman, 2002).

The attention model is portrayed as a weight matrix where the columns may represent the source sentence and the lines may represent the translation and each position (i, j) as an attention score evaluating importance (relationship) between two words (see Figure 2.13). For visual reasons, the authors usually plot the matrices using colors instead of numbers. In general, stronger colors represent larger or smaller numbers depending on the plot tool.

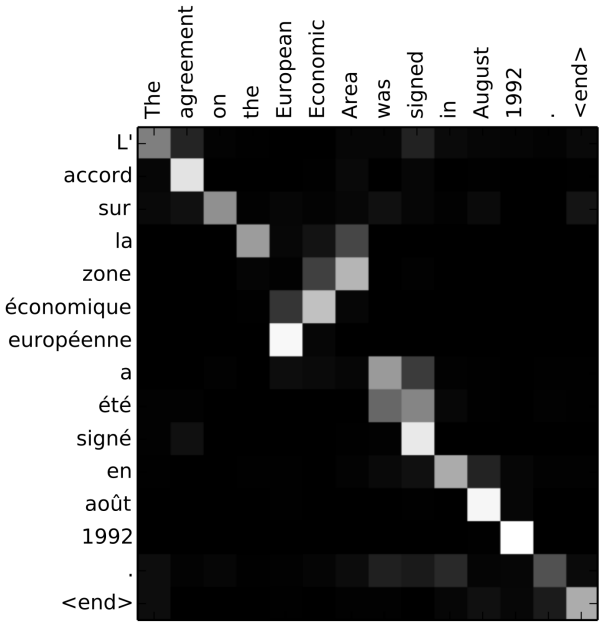


Figure 2.13: *Attention matrix for sentence translation (Bahdanau et al., 2014).*

In the text summary task, Figure 2.14, we can see the score (i, j) as a weight associated with each word that has a strong relationship with each word summary judgment. The columns represent the distribution of the entrance after generating each word.

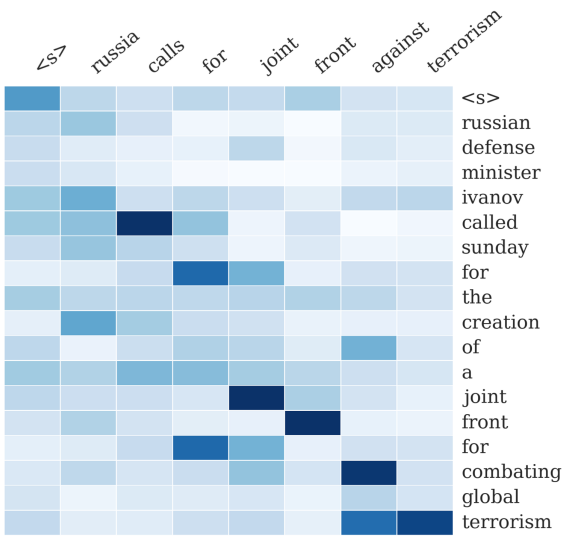


Figure 2.14: *Attention matrix for sentence sumarization (Rush et al., 2015).*

Other variations of the attention mechanism are proposed by (Luong et al., 2015). In that work, the authors develop two novel types of attention based models: a *global* approach that all source words are attended and a *local* one in which only a subset of source words are considered at a time. The first approach proposed by (Luong et al., 2015) is similar to the model of (Bahdanau et al., 2014) but it has a simpler architecture. The second approach can be viewed as an interesting blend between the hard and soft attention models proposed by (Xu et al., 2015): it is computationally less expensive than the global model or the soft attention; at the same time, unlike the hard attention, the local attention is differentiable, and, therefore, easier to be implemented and trained.

2.4 Ontologies and SPARQL

In this work, we are querying an ontology using the SPARQL query language, so it is important to understand the relationship between SPARQL and OWL Ontologies.

2.4.1 Ontologies

Ontologies are used to capture knowledge in a particular field of interest. To Gruber et al. (1993) ontology is "a formal specification of a conceptualization". Borst (1997) expands the definition by stating that this conceptualization can also be shared.

The Web Ontology Language (OWL) is a formal language to describe ontologies (Bechhofer et al., 2004), making them understandable by both humans and computers. Simple or complex concepts can be developed by using a wide range of operators that can offer OWL language.

The OWL class describe properties of a set of similar elements. The classes are organized hierarchically, this organization also known as taxonomy. Individuals are class instances which represent objects in a particular domain we are interested in. The properties are binary relations between individuals and this may be related to a data attribute, named given property (Datatype Property), or other individual named object property (Property Object).

Structure and Components

The ontologies contain a vocabulary for representing knowledge. This vocabulary is sustained for a predefined conceptualization, thus avoiding ambiguous interpretations. Ontologies also allow the sharing of this knowledge (Rabelo, 2011). Among the various elements that OWL ontologies have, here we will present only the most relevant for this work.

Individuals Individuals represent objects in the domain of interest. OWL does not use the *Unique Name Assumption (UNA)*. This means that two different names can refer to the same individual. For example, *Queen Elizabeth*, *The Queen* and *Elizabeth Windsor* may be references to the same individual. In OWL it should be stated explicitly whether individuals are the same or different from each other (Horridge et al., 2004).

Properties Properties are binary relationships between individuals, that is, the properties link two individuals. For example, the property *hasSibling* can link the individual *Matthew* to individual *Gemma*; or the property *hasChild* can link the individual *Peter* to individual *Matthew*. The property can also be reversed. For example, the inverse property of *hasOwner* is *isOwnedBy* (Horridge et al., 2004).

Properties can be limited to a single values: the *Functional Properties*. They can also be transitive and symmetric. The picture 2.15 shows properties that connect individuals (Horridge et al., 2004).

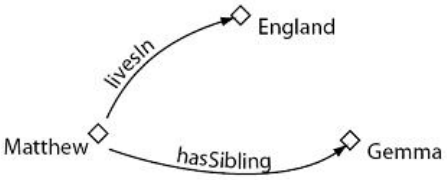


Figure 2.15: *Relation between properties and individuals (Horridge et al., 2004).*

Class OWL classes are sets that contain individuals. They are described formally (by mathematical descriptions) to present the requirements for participation in the class. For example, the class *Cat* may contain all cats individuals who are in the area of interest. Classes can be organized in superclass-subclass hierarchies, also known as taxonomies. Subclasses are specializations of its superclasses (Horridge et al., 2004).

Consider the class *Animal* and *Cat*: *Cat* can be subclass of *Animal*, so *Animal* is the superclass of *Cat*. This means that: All *Cats* are *Animals*; All members of the class *Cats* are members of the *Animal* class; Be a *Cat* means to be an *Animal*. A feature of OWL-DL is that the superclass-subclass relationship can be computed automatically by using a IE (Inference Engine). Figure 2.16 shows a representation of classes that contain individuals: The classes are represented as circles (Horridge et al., 2004).

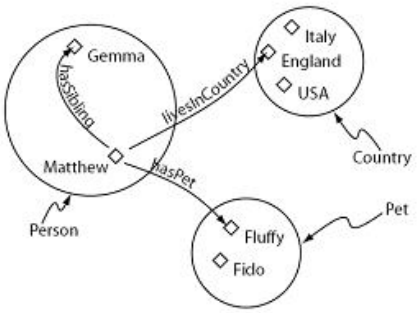


Figure 2.16: *Relation between classes (Horridge et al., 2004).*

2.4.2 SPARQL

SPARQL is an RDF query language, that is, a query for database language, able to retrieve and manipulate data stored in RDF format (Resource Description Framework). RDF is a labeled and directed data format used to represent the information in the web (Prud et al., 2006). SPARQL is

data-driven, for example, there is no inference into the language itself. It extracts only the ontology that is requested by the query.

Sintax

SPARQL has hundreds of elements that make up its syntax (Beckett, 2006), but we will describe the most important elements in the context of this work. The purpose is to convey a basic understanding of the construction and operation of SPARQL queries.

RDF Triples

RDF triple contains three components (Klyne et al., 2004):

- **Subject** : Can be individuals or ontology classes;
- **Predicate** : They are the relationships and can be object properties or data property, or some kind of feature set for version of RDF.
- **Object** : They are values of relationships, can be individuals, classes or primitive data.

RDF triple is conventionally written in the sequence: subject, predicate, object. The predicate is also known as the property of the triple (Klyne et al., 2004). The subject and the object are names for two "things" in the world, and the predicate is the name of a relationship between the two.

Sintax to triples The queries SPARQL are based in RDF triples, are lists of subject, predicate and object.

```
PREFIX p:<http://a.com/ontology#>
SELECT ?capital WHERE {
        p:texas_state p:has_capital ?capital
    }
```

In the SPARQL *script* above `rdfs` is an `uri` from ontology which is a resource identifier. `texas_state` is the subject, `has_capital` is the predicate and the variable `?capital` is the object which we want to find.

Types of query Structural types of consultation are (Beckett, 2006):

- `SELECT`: Used to extract raw values of a base RDF, the results are presented in a table;
- `CONSTRUCT`: Used to extract information from a RDF base and turn the result into a valid RDF;
- `DESCRIBE`: Used to extract a graph of a RDF base;
- `ASK`: Used to provide a simple boolean results for a query on an RDF base.

The `SELECT` structure can be used in combination with `DISTINCT`, which eliminates duplicate solutions (Prud et al., 2006). Another common function in SPARQL is the function `count` which is combined with `SELECT`. This combination is used to count results.

2.5 Context Free Grammars

A context-free grammar (CFG) is a set of recursive rewriting rules (or productions) used to generate patterns of strings. A CFG consists of the 4-tuple $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S_0)$, where:

- V is a finite set of *nonterminal* symbols;

- Σ is the vocabulary, consisting of a finite set of *terminal* symbols, $V \cap \Sigma = \emptyset$;
- $\mathcal{R}$ is a set of CFG rules, which we consider here as partitioned into two sets of rules: the terminal rules are of the form $X \to t$, $X \in V$ and $t \in \Sigma$; and the nonterminal rules of the form $X \to \mathbf{Y}$, $X \in V$ and $\mathbf{Y} \in V^*$;
- $S_0 \in V$ is the starting nonterminal.

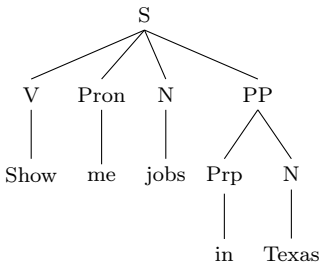
The language generated by context-free grammars are known as context-free languages (CFL). Different context-free grammars can generate the same context-free language. It is important to distinguish the properties of the language (intrinsic properties) from the properties of a particular grammar (extrinsic properties). The language equality question (do two given context-free grammars generate the same language?) is undecidable (De la Higuera, 2010, Sipser, 2006).

A context-free grammar provides a simple, mathematically precise mechanism for describing the methods by which natural language phrases are constructed from smaller blocks, capturing the "structure of these blocks" of sentences. Its simplicity makes formalism subject to rigorous mathematical study. Important features of natural language syntax, such as agreement and reference, are not part of context-free grammar, but the basic recursive structure of sentences, the way clauses nest within other clauses, and the way in which lists of adjectives and adverbs are swallowed up by nouns and verbs, is described (Charniak, 1997).

Parse tree or derivation tree or syntax tree is an ordered, rooted tree that represents the syntactic structure of a string according to some context-free grammar. For example, given the grammar:

S	→	V Pron N PP
PP	→	Prp N
V	→	Show
Pron	→	me
N	→	jobs
Prp	→	in
N	→	Texas

The sentence "Show me jobs in Texas" has the following parse tree:



Context-free grammars are simple enough to allow the construction of efficient parsing algorithms that, for a given string, determine whether and how it can be generated from the grammar, Earley (Earley, 1970) and CYK (Chappelier et al., 1998) are examples of these algorithms. Both Earley and CYK algorithms executes in cubic time in the general case $O(n^3)$.

Chapter 3

Related Works

In this chapter we present research literature related to different aspects of our work. In the first section we will talk about related works that seek to translate from natural language to SPARQL in a general way. In the second section we discuss other works that seek to improve vector representation for a formal language. In the third and last section of this chapter, we are dedicated to discuss works that aim to implement semantic parsing with neural networks, giving emphasis to structured predictions.

3.1 Translation from Natural Language into SPARQL and Query Ontologies

The translation from natural language into SPARQL has received a lot of attention when the semantic parsing task is related to Question Answering Systems (Allam and Haggag, 2012), because there is currently a series of repositories whose query language is based on SPARQL (Auer et al., 2007, Bizer et al., 2008). In chronological order, we summarize some works for this purpose.

3.1.1 Querix - Query ontologies based on clarification dialogs

The Querix (Kaufmann et al., 2006) employs a statistical approach. Given a query, the system consists of parsing, removing important elements and then looking for triples that are related to the elements of the query. Querix works as a component and can be adapted to any application. It is based on clarification of dialogues, so when there is ambiguity the system asks the user to decide.

This work was one of the first to attempt to transform natural language into SPARQL. Although its somewhat rudimentary approach has achieved interesting results, unfortunately it has been discontinued.

3.1.2 SQUALL - A logical approach

Another related work is the SQUALL (Ferré, 2012) that takes a logical approach. Natural language is transformed into an intermediate language based on Montague's grammar. Definitions for each λ-term are specified by associating the semantics of the intermediate language with SPARQL. SPARQL is generated by applying the definitions. The work has an interesting implementation but does not present an evaluation.

3.1.3 Translating from Linguistic Analysis

The work in (AlAgha, 2015) associates phrases in natural language with RDF triples, as in our approach. Through a linguistic analysis, their model extracts relations and associates them to triples. They also generate SPARQL scripts, however using first an intermediate format. Then, with the help of an ontology, a SPARQL query is generated after identifying the targets and modifiers of the query.

This work was developed using Arabic sentences as input. Despite having a very different approach the idea is basically the semantic compositionality of elements of the language and the relation of these elements between languages. For example, a sentence in natural language can be composed of nominal phrases (NP), verbal phrases (VP) among others. The authors associate these elements with elements of the SPARQL language, for example the RDF triples (see Section 2.4.2).

3.1.4 SPARQL as a Foreign Language

In (Soru et al., 2017), the authors propose a concept called Neural SPARQL Machines, end-to-end deep architectures to translate any natural language expression into sentences encoding SPARQL queries.

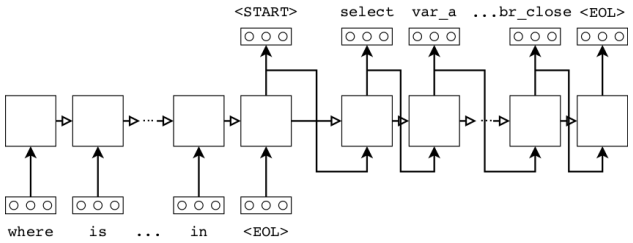


Figure 3.1: *Architecture for Neural SPARQL Machines (Soru et al., 2017)*

In the Figure 3.1, we can see LSTM architecture for sequence-to-sequence learning of a machine translator from natural language to SPARQL. The left side is the encoder, whereas the right side shows the decoder. In this work, the authors use the Neural Machine Translation (Bahdanau et al., 2014) incorporating SPARQL as a foreign language, as can be seen in Figure 3.1, the left network is the natural language encoding, and the right side is meta SPARQL decoding. The model also has post processing to make SPARQL suitable for use. According to the authors, the model *Neural SPARQL Machines* is a promising approach for Question Answering on Linked Data, as they can deal with known problems such as vocabulary mismatch and perform graph pattern composition.

In this work there is no syntactic guarantee for the produced SPARQL, and initialization of all vectors is random with *end-to-end* learning.

3.2 Vector Representation to Formal Languages

To the author's best knowledge, there is no paper dedicated to improve the vector representations of formal languages. The most common approach when it comes to encoder decoder architectures aims to train the lexicon vectors themselves along with encoding decoding networks (end-to-end). Although interesting results may stem from this approach, it is well known that a model when initialized with vectors pre-trained achieves better results (Bengio et al., 2003).

3.3 Structured Prediction in Neural Semantic Parsing

After some works have consolidated the LSTM Encoder-Decoder architecture as being powerful for translation problems (Bahdanau et al., 2014, Cho et al., 2014a,b, Luong et al., 2015) the adequacy of this concept to semantic parsing problems was expected. However, well-formed sentences as logical expressions were not guaranteed by the traditional model.

Between the years of 2016 and 2017 many semantic parsing models based on neural networks able to making structured predictions emerged (Goldman et al., 2017, Jia and Liang, 2016, Krishnamurthy et al., 2017, Locascio et al., 2016). One of the first works incorporating the encoder decoder architecture with neural attention was (Dong and Lapata, 2016) where they also bring a proposal for structured prediction. Let's summarize the main semantic parsing works using neural networks and with structured prediction.

3.3.1 Language to Logical Form with Neural Attention

Dong and Lapata (2016) present a general method based on an attention-enhanced sequence-to-sequence model. They encode input sentences into vector representations using recurrent neural networks, and generate their logical forms by conditioning the output on the encoding vectors. The model is trained in an end-to-end fashion to maximize the likelihood of target logical forms given the natural language inputs.

The authors also present an architecture called sequence-to-tree. This architecture assisted in the generation of well-formed structures with the use of a concept named hierarchical tree decoder.

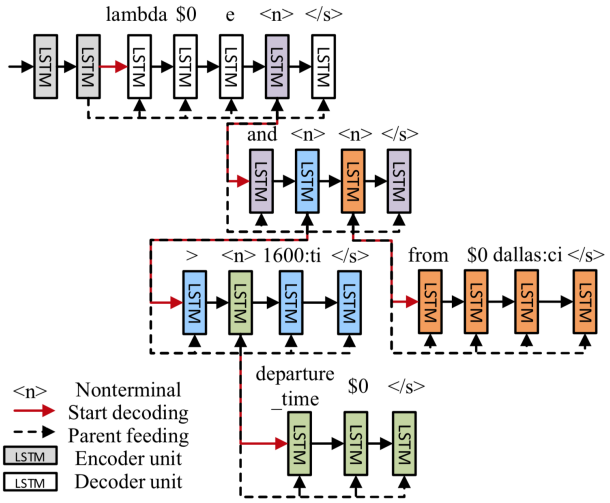


Figure 3.2: *Sequence-to-tree (SEQ2TREE) model with a hierarchical tree decoder*

As we can see in Figure 3.2, there is a non-terminal element $< n >$ which indicates subtrees. Let q be the input sentence and a the corresponding formal representation. After encoding input q, the hierarchical tree decoder uses recurrent neural networks to generate tokens at depth $+1$ of the subtree corresponding to parts of logical form a. If the predicted token is $< n >$, we decode the sequence by conditioning on the nonterminal's hidden vector. This process terminates when no more nonterminals are emitted. In other words, a sequence decoder is used to hierarchically generate the tree structure.

3.3.2 Sequence-based structured prediction for semantic parsing

In the work of (Xiao et al., 2016), the authors develop some improvements departing from (Wang et al., 2015). The grammar used in (Wang et al., 2015), along with generating logical forms, generates so-called *canonical forms* (CF), which are direct textual realizations of the Logical Form (LF) that, although they are not *natural* English, transparently convey the meaning of the LF.

Based on this grammar, the authors explore three different ways of representing the LF structure through a sequence of items. The first one, and simplest, consists of just linearizing the LF tree into a sequence of individual tokens; the second one represents the LF through its associated CF, which is itself a sequence of words; and the third one represents the LF through a derivation sequences (DS), namely the sequence of grammar rules that were chosen to produce this LF.

In the Figure 3.3, we can see example of Natural Language (NL), Canonical Form (CF), Logical form (LF), Derivation Tree (DT) and Derivation Sequence (DS) used in (Xiao et al., 2016).

NL: article published in 1950
CF: article whose publication date is 1950
LF: get[[lambda,s,[filter,s,pubDate,=,1950]],article]
DT: s0(np0 (np1 (typenp0), cp0 (relnp0, entitynp0))
DS: s0 np0 np1 typenp0 cp0 relnp0 entitynp0

Figure 3.3: *Dataset tuple example*

The authors predict the LF via LSTM-based models that takes as input a Natural Language (NL) question and map it into one of the three sequentializations. For example, in Figure 3.4 a derivation tree from which is extracted leftmost derivation sequence "s0, np0, np1, typenp0, cp0, relnp0, entitynp0".

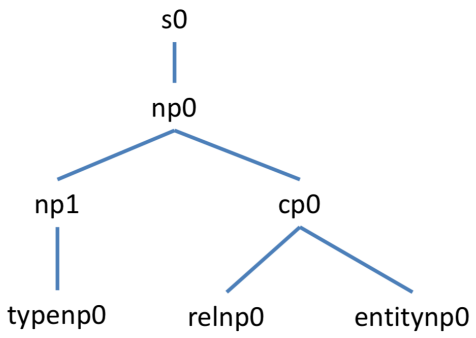


Figure 3.4: *Example of tree on (Xiao et al., 2016)*

In summary, in this work, for each of the structured generation ways there are two important parts. The first and most challenging is to map sentences in natural language to their canonical form, similar to the problem of finding paraphrases; the second part is to transform the canonical forms into logical forms, however this has already been obtained at work (Wang et al., 2015), using the G grammar. The define Clause Grammar (DCG) (Pereira and Warren, 1980) is the formalism used to transform the G grammar into the logical form.

3.3.3 Neural Generation of Regular Expressions

On the (Locascio et al., 2016) the authors propose the use of the LSTM encoder decoder architecture to translate from natural language sentences into regular expressions. The authors also propose a methodology for collecting a larger corpus of regular expression, natural language pairs. The data generation process has two steps. In the first one they generate regular expression representations from a small manually-crafted grammar. This grammar was extracted from the analysis of a small dataset used in (Kushman and Barzilay, 2013). Every grammar rule has associated verbalizations for both regular expressions and language descriptions. In the two step, the authors rely on crowd-sourcing to paraphrase these rigid descriptions into more natural and fluid descriptions. Using this methodology the authors have constructed a corpus of 10.000 regular expressions, with corresponding verbalizations. According to the authors, the results obtained for task of natural language translation in regular expression surpass the state of the art.

3.3.4 Neural Semantic Parsing with Type Constraints

According to (Krishnamurthy et al., 2017), one of the limitations in (Dong and Lapata, 2016, Jia and Liang, 2016, Locascio et al., 2016) is the inability to deal with type constraint, which is essentially important for a language that will be submitted to a compiler. One of the innovations of these authors was to add a grammar to the decoder capable of generating well-typed logical forms.

The work (Krishnamurthy et al., 2017) presents a semantic parsing model to answer questions composed in semi-structured tables of Wikipedia. In addition to ensuring well-structured prediction, the authors also have a solution to the problem of entity identification. The authors depart from the standard LSTM encoder decoder architecture. The input of the network are questions in natural language and a context to which they must be answered. The encoder is a bidirectional LSTM network, where natural language vectors are concatenated with vectors that represent entity link. The authors create a special grammar called "Type-Constrained Grammar", with this grammar it is possible to maintain a state at each step of the decoder. Briefly, decoder is an LSTM that outputs a distribution over grammar actions using an attention mechanism over the encoded question tokens.

3.3.5 Differences between our approach

While in the earlier works the predicted language is generally a logical form based on lambda expressions, we make use of another very well-founded formalism, the context-free grammars. Although all these works will be important contributions in Neural Semantic Parsing with structured prediction, in fact, our approach is the least onerous from the point of view of cross-domain since the others suggest the creation of special grammars. In addition to common elements of supervised computational learning, we use a context-free grammar capable of parsing the target language. Although we do not have a restriction on our grammar, we can realize during the experiments that simpler grammars, without syntactic ambiguity and without recurrent rules, perform better.

Chapter 4

From Natural Language to SPARQL

The purpose of our work is to semantically parse English into SPARQL employing methods that do not rely on handcraft-rules, high-quality lexicons, manually-built templates or other handmade complex structures. We chose the English language by ease of finding datasets in this language. As seen in Section 3.3, all approaches related to neural networks and structured prediction up to the present moment leave something to be desired about this premise. In our structured prediction model, the only new input we need is a context-free grammar that can parse our target language. Our improvement of target language vectors starts with well known methods. In this sense, we believe that we are actually following the previously established premise of using the least amount of manually constructed resources. This chapter is organized as follows. In Section 4.1, let us present our contribution in relation to the lexical part of semantic parsing. In Section 4.2, we present the contribution of improvement in the syntactic aspect.

4.1 Improving SPARQL vector representation with neural attention

The literature on Neural Semantic Parsing lacks works about vector representation of formal languages as a target language. In general, in every work about Neural Semantic Parsing, the initialization of the vectors representing the lexicon was random with adjustments according to the training. Although this approach is efficient, several other studies in machine learning mention that the results improve with pre-trained vectors (Bengio et al., 2003, Tomáš, 2012).

In the case of natural languages, there is a very large volume of data available on the web that reflects the structure of a language in a more general way, for example wikipedia, which makes available its entire knowledge base through texts (Auer et al., 2007). For natural languages, there are solid methods for generating word embeddings from text. In formal languages, the construction of queries are dependent on the modeling of the information base so it is difficult to acquire a large number of examples.

The natural question that arises is "how do we find better vectors to represent formal languages?". We have seen in the literature on neural attention mechanism a way to solve, at least in an approximate form, the problem of matching between terms of two different languages. Our proposal aims to use natural language vectors also to represent SPARQL words. The procedure to obtain the refinement of the SPARQL vectors occurs in two phases:

- We do a neural semantic parsing training using a SPARQL version of the Geo880 corpus (see Section 5.1) and LSTM Encoder Decoder with Neural Attention. In this training, the vectors of English words are extracted from the Glove (Pennington et al., 2014) and the vectors of the SPARQL part are initialized randomly into a Gaussian distribution. In this phase, we generated a *Global Alignment Matrix* using Neural Attention;
- We apply matching using the *Global Alignment Matrix* and at the end we get the new set of vectors for SPARQL.

The results of this work have been published as "Semantic Parsing Natural Language into SPARQL: an LSTM Enconder-Decoder Neural Net Approach" at ENIAC 2017 having received a prize of the best paper of the event.

Global Alignment Matrix Let $S = \{s_1, s_2, \ldots, s_m\}$ be the set of words in the source language (in our case, the natural language) and $T = \{t_1, t_2, \ldots, t_n\}$ be the set of words in the target language (in our case, the SPARQL). The Global Alignment Matrix Λ is a $|S|$ by $|T|$ matrix whose value in position (i, j) represents the level of alignment that the word $s_i \in S$ has with the word $t_j \in T$.

4.1.1 Vocabulary matching using global alignment matrix

We propose an association between each word from the target language with another one in the source language with the *greatest correlation*.

Let S and T be a source and a target language vocabulary, respectively, and let $\Lambda \in \mathbb{R}^{|S|\times|T|}$ be the global alignment matrix. Let $s_{i^*} \in S$ be the word that possesses the *greatest correlation* with the word $t_j \in T$. We can define it as

$$\max_{i=1,\ldots,|S|} \Lambda_{ij},$$

and the word s_{i^*}, with

$$i^* = \operatorname*{argmax}_{i=1,\ldots,|S|} \Lambda_{ij}.$$

Suppose that Figure 4.1 illustrates a global alignment matrix. To improve visualization, it was plotted with colors instead of numerical values: the darker a cell is the bigger is the correlation between the terms row and column. For example, in the Figure 4.1, the target word **?texas** has *greatest correlation* with source word **texas**.

To exemplify the matching procedure, notice that the target word **p:has_city** matches the source language word **cities**. In fact, for the column corresponding to the target word **p:has_city**, the darkest cell connects it with the source language word **cities**. That is, the word **p:has_city** has the *greatest correlation* with the word **cities**. Based on this table, we have built a translation dictionary from target to source. This dictionary could have several practical uses, for example, the lexical representation of the target. It will be further detailed in the next section.

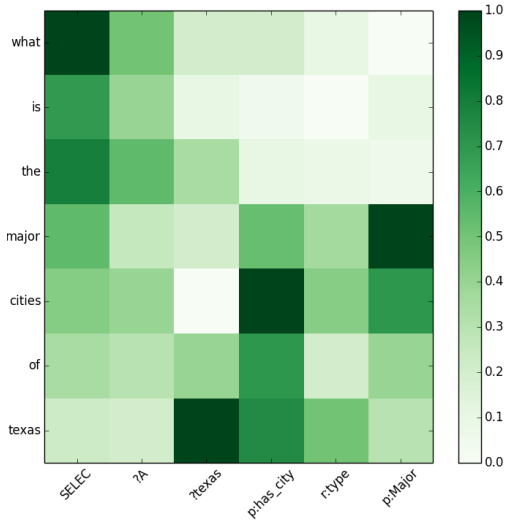


Figure 4.1: *Neural attention matrix example*

4.1.2 Lexical representation

To represent the lexicon of natural language, we employ the model developed in Pennington et al. (2014). However, in the case of SPARQL terms, we performed several experiments to analyze

different representations. We first describe the vector representation generation methods for the lexicon we use. The choice of the following methods is due to their popularity:

RANDOM: This representation was randomly generated using a normal distribution as a kernel with values from -1 to 1. This approach was chosen in order to kick off the representation like a baseline and even if it could not capture the relationship between terms, its performance was not so low when compared to more sophisticated approaches.

TF-IDF/PCA: We use the target language sentences and generate a term document matrix. In this matrix, we consider each query a document and each word in the target language a term. After generating the term-document matrix we apply TF-IDF Aizawa (2003). At the end we did a size reduction using Principal Component Analysis (PCA) Jolliffe (2002). Our reduction was from 880 to 300 dimensions.

W2V10: This approach is based on Mikolov et al. (2010), in which the author proposed the generation of a vector representation of words based on recurrent neural networks, using only text as input. We use the target language sentences as input text to generate a vector representation.

OUR-APP: Our approach is focused on using the same set of vectors to represent both source and target language vocabulary. What we do is a match between the terms of the source language and the target language seeking a match between the two vocabularies. The matching process is possible with the alignment table generated by the neural attention mechanism.

First using a random version to represent the target's lexicon, then we train and generate an alignment table, Figure 4.1. Then, we use the **Matching** (see section 4.1.1) to associate the lexicon of the source language with the target language. We use a heuristic to associate all elements of one vocabulary with another. SPARQL words are similar to English words, for example, "SELECT", "FILTER", "ORDER", were directly associated with their respective correspondents in English. The other terms such as subject, predicate and object of the triples, are solved with the matching procedure.

Note that we do not deal with the "out of vocabulary" situation (OOV), nor with lexical disambiguation. The lexical ambiguities are assumed to be codified inside the attribute vectors in the pre-processing; furthermore, approximately 1% of the words in the vocabulary are lexically ambiguous, allowing us to safely ignore such effects.

In Section 5.3 the results show that for all tests we obtained better results when the vectors were initialized with our approach. Another problem we noticed during our tests is that the SPARQL queries was not always grammatically correct. How to deal with the generation of non-grammatical (incorrect) SPARQL expressions will be dealt in the next section.

4.2 Syntactic Assurance to Target Sentence

Although the Neural Semantic Parsing models take advantage of the great efficiency of deep neural networks, in their standard edition, they are not able to predict sentences with structure or syntax guarantees. In this section, we present a refinement of the encoder-decoder architecture capable of guaranteeing structured prediction. We are primarily concerned with generating a target language that obeys a given context-free grammar. We are also concerned with the construction of training corpora for the case where the target language must obey a given CFG.

A CFG is a 4-tuple $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S_0)$, where V is a finite set of *nonterminal* symbols; Σ is the vocabulary, consisting of a finite set of *terminal* symbols, $V \cap \Sigma = \emptyset$; $S_0 \in V$ is the starting nonterminal; and $\mathcal{R}$ is a set of CFG rules, which we consider here as partitioned into two sets of rules: the terminal rules are of the form $X \to t$, $X \in V$ and $t \in \Sigma$; and the nonterminal rules of the form $X \to \mathbf{Y}$, $X \in V$ and $\mathbf{Y} \in V^*$.

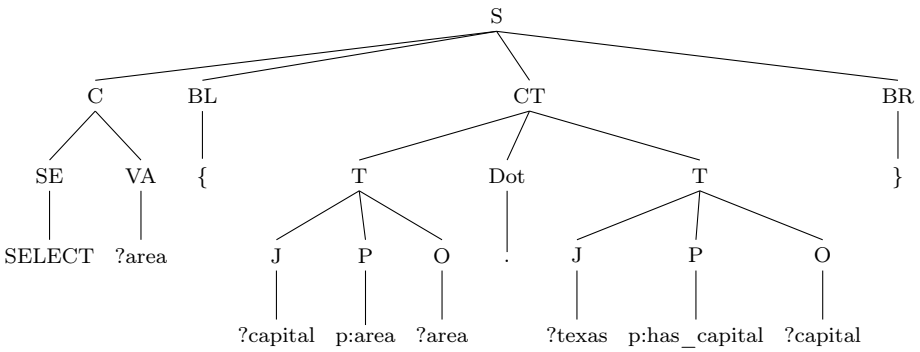
The parsing of an input sentence $x_1, \ldots x_n$ generates a *parsing tree*, whose leaf is the input sentence and whose internal nodes are nonterminals. We say that an occurrence of a nonterminal X in a parsing tree dominates the sequence $x_t, \ldots, x_{t+r}$ if the subtree rooted in X has leaves $x_t, \ldots, x_{t+r}$. The initial symbol S_0 always dominates the whole input sentence.

4.2.1 SPARQL Grammar

To illustrate our method we restrict ourselves to a fragment of SPARQL queries given by the grammar below. The first four rules are the proper grammatical rules; the remaining ones are the lexical rules which generate the language terminals. In this example we analyze the query `SELECT ?area { ?capital p:area ?area . ?texas p:has_capital ?capital }`:

S	→	C BL CT BR
C	→	SE VA
CT	→	T Dot T
T	→	J P O
Dot	→	.
BL	→	{
BR	→	}
SE	→	SELECT
VA	→	?area
J	→	?capital
P	→	p:area
O	→	?area
J	→	?texas
P	→	p:has_capital
O	→	?capital

This grammar generates the following parsing tree:



This sentence in SPARQL corresponds to the following sentence in natural language: “what is the area of the capital of Texas?”.

4.2.2 Encoder CFG-Decoder Model

In our approach we add some elements to the traditional encoder-decoder model in order to improve the training and generation phases. The method takes as input a natural language sentence and a SPARQL grammar and gives as output a query generated by that grammar. It is important to notice that the query is generated in depth-first rightmost way, so that the actual query at the leaves of the tree is generated backwards, starting from its rightmost symbol and progressing leftward to the first symbol. In this way, the tail of a query is available as a “context” $\chi = y_{s-r+1}, \ldots, y_{s-1}, y_s$ during query generation, and it will be used both for encoding and decoding. In both training and generation, the context χ is initially empty and grows during the execution of the process.

The encoder CFG-decoder models consists of a set of pairs of encoder-decoder networks, one for each nonterminal, $\{\langle \mathcal{E}_X, \mathcal{D}_X \rangle | \, X \in \mathcal{V}\}$. During the prediction process, there is a *controller mechanism* that, given a context $\chi = y_{s-r+1}, \ldots, y_{s-1}, y_s$, chooses a nonterminal $X \in \mathcal{V}$ for expansion, such that in the output parsing tree X will dominate $y_u, \ldots, y_{s-r}$.

The controller is simply a stack $\mathcal{S}$ of nonterminal symbols, and the choice of which nonterminal is expanded is obtained simply by popping the top nonterminal in $\mathcal{S}$, and the decoder can push new items to $\mathcal{S}$ according to the rules of the input grammar. Initially, $\mathcal{S}$ contains the single nonterminal S_0, the grammar initial nonterminal, and $\chi = \varepsilon$, the empty string.

At each prediction step, a nonterminal X is popped from $\mathcal{S}$. The encoder $\mathcal{E}_X$ receives as input the whole natural language sentence W and the current context χ, and generates the encoding c. The decoder $\mathcal{D}_X$ then receives as input c and the context χ, and it has to choose from the grammar rules of the form $X \to \mathbf{Y}_i$ which one will be expanded.

There may be k candidate rules headed by X, say, $X \to Y_{i1}Y_{i2}\dots Y_{im_i}\bullet$, $1 \le i \le k$, where $\bullet$ is the stop element in the rule's tail. The decoder's task is to choose one specific rule tail among the possible ones, by generating a string that maximizes the probability

$$\rho_+ = P(Y_{it}|Y_{t-1}, ..., Y_1, \chi, c_X). \tag{4.1}$$

Each decoding step outputs a symbol Y_j, narrowing down which rule can be chosen. The main idea behind this is to eliminate all candidates whose j-th symbol is different from Y_j. At the end, a single rule is chosen. The strategy to choose this rule is as follows. Let $m = \max_{1\le i\le k}\{m_i\}$ and suppose we are in the ℓ-th prediction step. If $\ell > m$, we are done, since all rules are distinct and at each prediction step j we eliminate all the rules with j-th nonterminal symbol different from Y_j. Otherwise, given the probability ρ of Y_{j-1}, if there is any rule with size $j-1$ and $\rho_+ < \rho$, we choose the rule X that finishes with Y_{j-1} and we are done. If the rule chosen is a terminal $X \to t$, the context is updated by concatenating t to the front of the current context. If a nonterminal rule is chosen $X \to Y_1 \cdots Y_\ell$, the tail is pushed to the stack $\mathcal{S}$, so that its top now contains Y_ℓ. The prediction process then proceeds until $\mathcal{S}$ becomes empty. We illustrate this process in Figure 4.2.

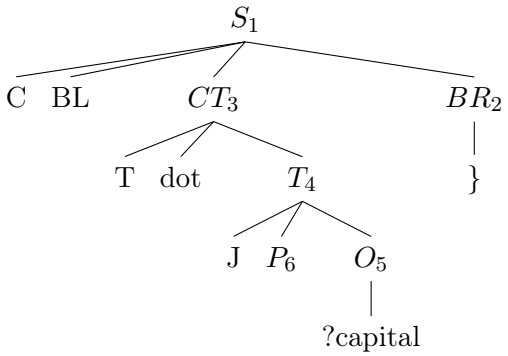


Figure 4.2: *Partial generation*

In the Figure 4.2, the subindex in each symbol means the order in which each rule is expanded. In this illustration, the next rule to be expanded is the one headed by P. The input sentence and the context $\chi =$ } ?capital is passed to the encoder $\mathcal{E}_p$ to generate the encoding c, that will be used next by the decoder $\mathcal{D}_p$.

The Training Algorithm

During training we assume that we have a tree in which the leaves contain the target query. The decoder generates a tree in a top down, depth-first fashion. The actual query is generated right to left, that is, the last symbols of the output query sentence are produced first. This strategy allows us to consider the output sentence tail as a context used during query generation; that tail is also used in training of the encoder.

We perform the training phase using a supervised dataset that consists of a list of pairs (W, q), where W is an English sentence and q is the corresponding SPARQL. Algorithm 1 describes the

training process and is called for each pair ⟨sentence, query⟩ of the training set for $\mathcal{E}$ and $\mathcal{D}$ weight readjustment.

Algorithm 1: Encoder CFG-Decoder training

Input. A sentence W and its corresponding SPARQL query q. A SPARQL context-free grammar $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S_0)$. The context size r.

Output. The LSTM encoder-decoder trained model.

Step 1. **Initialization.** Generate the parser tree $\mathcal{G}_q = (\mathcal{V}, \Sigma, \mathcal{R}_q, S_0)$ from SPARQL query Q. Let X be the nonterminal under expansion, $X \leftarrow S_0$; let c the current encoding, $c \leftarrow \varepsilon$; χ is the decoding context, $\chi \leftarrow \varepsilon$. Consider an stack or nonterminals, $\mathcal{S} \subset \mathcal{V}_q$, initially empty.

Step 2. **Training.**

Step 2.1. Let $r \in \mathcal{R}_q$, $\mathcal{E}_X$, and $\mathcal{D}_X$ be the rule, the encoder, and the decoder associated with the nonterminal X, respectively.

Step 2.2. If $= X \to Y_1 \cdots Y_m$ is a nonterminal rule,

Step 2.2.1. Push $Y_1 \cdots Y_m$ to $\mathcal{S}$, in the same order they appear in r.

Step 2.2.2. Perform a forward step in $\mathcal{E}_X$ using the sentence W and the context χ, generating the encoding c.

Step 2.2.3. Perform a forward step in $\mathcal{D}_X$ using encoding c, unfolding its output m times to obtain $Z_1 \cdots Z_m$; compute the error with respect to $Y_1 \cdots Y_m$ and store it for the backward step.

Step 2.3. If $r = X \to t$ is a terminal rule, concatenate t to χ. If χ has size greater than r, remove the rightmost character from χ.

Step 2.4. Perform a backward step in $\mathcal{D}_X$ and $\mathcal{E}_X$.

Step 3. If $\mathcal{S}$ is not empty, let X be the top item; pop X from $\mathcal{S}$ and go to Step 2. Otherwise, return $\mathcal{E}$ and $\mathcal{D}$.

The main element used in the training phase is a *parser tree* $\mathcal{G}_q$. We generate this parser tree from q using the SPARQL grammar we defined in Section 4.2.1 and an Earley parser Earley (1970). This parser analyzes the query q and generates $\mathcal{G}_q$ as a list of rules $\mathcal{R}_q$ used to generate Q.

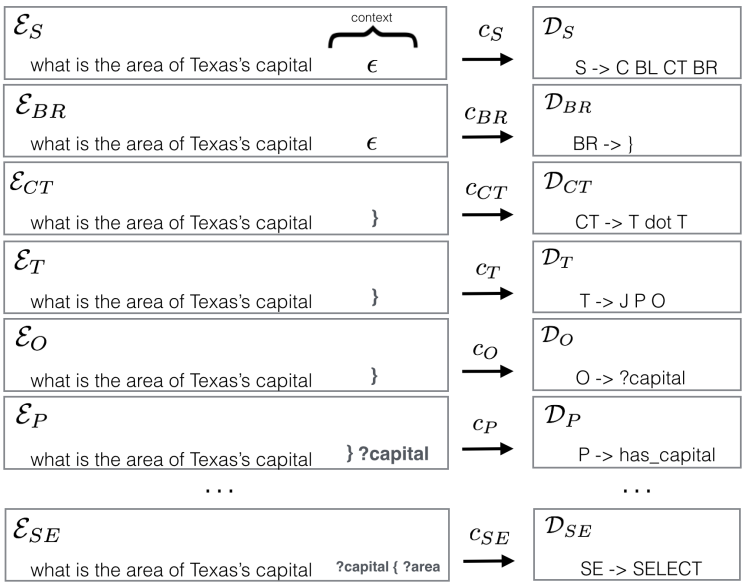


Figure 4.3: *CFG Decoder scheme (3-context).*

The encoder-decoder model consists of a set of n encoders and n decoders, where n is the number of nonterminals in the SPARQL grammar. This means that we have a specialist encoder and decoder for each nonterminal symbol.

In the query prediction phase, as we allow for more than one rule with the same nonterminal at the head, we need a mechanism to choose an appropriate rule for a given nonterminal symbol. For that, we added the notion of a *context*, χ, which provides the decoder with information based on which choice can be made. The context consists of s terminal symbols in the generated query; as the query is generated from right to left, it consists of the latest s terminals generated. We illustrate this feature in Figure 4.3, the context is the last terminal symbol from the right-hand side of the input SPARQL q, i.e. $s = 3$.

In Figure 4.3 we use the notation c_X, where $X \in V$ is a nonterminal symbol to indicate that each neural network generates a specialized c encoding to $X \to \mathbf{Y}$ rule, where X is a symbol of the head and $\mathbf{Y}$ is a set of symbols that compose the tail.

Prediction

The prediction (also called generation) process is described in Algorithm 2. It follows the description of the runtime algorithm given above.

Algorithm 2: Encoder CFG-Decoder prediction

Input. A sentence W, an LSTM encoder-decoder trained model, and a SPARQL context-free grammar $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{R}, S_0)$. The context size α.

Output. A SPARQL query q.

Step 1. Consider an empty stack $\mathcal{S} \subset \mathcal{V}$. Let X be an empty string, the current nonterminal $X \leftarrow S_0$, encoding $c \leftarrow \varepsilon$ and context $\chi \leftarrow \varepsilon$.

Step 2. Let $\mathcal{R}_X \subset \mathcal{R}$, $\mathcal{E}_X$, and $\mathcal{D}_X$ the set of rules, the encoder and the decoder associated with the nonterminal X, respectively.

Step 3. Compute an encoding c using the encoder $\mathcal{E}_X$ with the sentence W and the context χ. Let j be the latest position generated by the decoder, $j \leftarrow 0$, and let ρ be the probability of the symbol generated at j, $\rho \leftarrow 0$.

Step 3.1. Let ℓ be the size of the largest rule $r \in \mathcal{R}_X$. If $\ell < j$, go to Step 4. Otherwise, let $\mathcal{L}$ be a list containing the distinct j-th symbols from the rules in $\mathcal{R}_X$.

Step 3.2. Check which rule $X \in \mathcal{L}$ has the largest probability ρ_+ using the decoder $\mathcal{D}_X$ with the encoding c. Let v be the j symbol of X.

Step 3.3. If there is any rule in $\mathcal{R}_X$ with size $j - 1$ and $\rho_+ < \rho$, remove all rules from $\mathcal{R}_X$ with size greater or equal than j and go to Step 4.

Step 3.4. Remove from $\mathcal{R}_X$ rules with size less than j and whose j-th symbol is different from v.

Step 3.5. Let $\rho \leftarrow \rho_+$, $j \leftarrow j + 1$, and go to Step 3.1.

Step 4. Let r be the remaining rule in $\mathcal{R}_X$. If r is a nonterminal rule, add to $\mathcal{S}_X$ the nonterminals from the tail of r, in the same order they appear in the rule. Otherwise, let $\sigma \in \Sigma$ be the symbol from the tail of r. Concatenate σ in the beginning of c and Q. If c has size greater than α, remove the rightmost character from c.

Step 5. If $\mathcal{S}_X$ is not empty, let p be its the top item; remove p from $\mathcal{S}_X$ and go to Step 2. Otherwise, return Q.

Improvements in the model

Neural Attention When we analyze the case of the example shown in Section 4.2.1, the training pair:

source	what is the area of the capital of Texas?
target	SELECT ?area { ?capital p:area ?area . ?texas p:has_capital ?capital }

We can state intuitively that rule $J \rightarrow$?texas is related with the word **Texas**. Thus, it seems inappropriate to associate an entire sentence with only one rule of grammar. For this reason, we adopted the mechanism of neural attention, the effect of this is that we reinforce an association between that part of the sentence related to the rule that we need to learn and predict.

On Figure 4.4 we show an alignment matrix obtained during our training. We add to our model a layer of attention similar to that used in the work (Dong and Lapata, 2016).

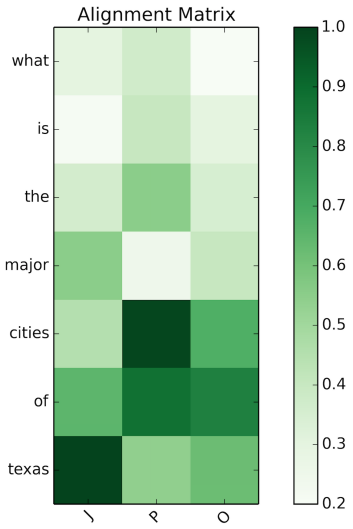


Figure 4.4: *Neural attention matrix example*

As we can see in Figure 4.4, in the source sentence part (sentence in matrix column) we do not show the context to simplify the visualization.

Context Representation We define as context element in our proposal, χ context, it is intended to capture the derivation history of the rule that we are expanding. It is a challenge to represent context effectively. In the future work section we highlight this challenge.

Chapter 5

Evaluations, Results, and Discussion

We compared our methodology with related work using the two well-known datasets to be described next. Here we first define metrics for comparing different implementations and then describe an adaptation of the datasets to SPARQL target language; we plan to make that adaptation, as well as our implementation freely available. We also discuss the occurrence of syntactic errors in some of the implementations used for comparison; note some specifics of neural network settings; and finally comment on comparisons of our work with different approaches.

The metrics used for comparison in the experiments are:

$$\text{Accuracy} = \frac{\#\text{ of correctly translated queries}}{\#\text{ total of queries}}$$

$$\text{Syntactical Errors (S.E.)} = \frac{\#\text{ of queries with syntactical errors}}{\#\text{ total of queries}}$$

In Section 5.2.1, we describe Syntactical Errors.

5.1 Datasets

Our experiments were conducted using two traditional datasets:

- Geo880, a set of 880 queries to a database of U.S. geography. The data were originally annotated with Prolog style semantics as the target language, which we manually converted to equivalent statements in SPARQL. On the official web page of the Geo880 dataset at the University of Austin in Texas one can find a database consisting of Prolog facts, **geobase**, and natural language questions directed at this domain, **geoquery880**. First we created an OWL ontology based on **geobase** and then, for each query in natural language in **geoquery880**, we wrote a corresponding SPARQL query.
- Jobs640, a set of 640 queries to a database of job listings. This dataset contains computer-related job postings, such as job announcements, from the USENET newsgroup austin.jobs. We created an OWL ontology based on jobs640 and then we wrote a corresponding SPARQL query to each query.

Previous work Dong and Lapata (2016), Tang and Mooney (2001), Zettlemoyer and Collins (2005) described these data sets. Both the ontology and the set of questions can be found in our repository https://github.com/enc-cfgdec/evaluation-data .

5.2 Settings

Finding better parameters and hyper-parameters for neural networks is always a very costly task. The first rounds with the neural network (pre-tests) served to find the best parameters for the neural network.

- Learning rate: During the pre-test, we used two different learning rates. In the first one, we maintained the learning rate at 0.5. In the second pre-test, we varied the learning rate, starting at 1 decreasing 5% at each iteration (epoch). The second case provided the best results;
- Epoch: In the tests we chose to use 100 epochs, as this was the best result found in our pre-tests for hyperparameters;
- Hidden dimension: With regard to the number of hidden layers, we used a series of pre-tests with 100, 200 and 400 hidden layers. We continued testing with the 4 different dimensions;
- Input dimension: We used three different input dimensions, 100, 200 and 300 but the best results were obtained with vectors of size 300.

5.2.1 Syntactical Errors

For the purpose of these experiments, a generated SPARQL statement contains a *syntactical error* if it cannot be processed by SPARQL interpreter like Protégé [1] due to syntactical formation[2]. In general, there are several types of syntactical errors that can be generated, such as not closing an open brackets or even a type mismatch between a function and a variable provided as an argument.

5.3 Results on SPARQL vector representation

In Tables 5.1 and 5.2 we show the experiments performed using our method to represent the target lexicon. In this experiments, we use two implementations of the LSTM encoder decoder neural network with the configuration described in Section 5.2. One implementation with neural attention and another without. To represent the lexicon vocabulary, we use vectors with each of the methods presented in Section 4.1.2: RANDOM, TF-IDF/PCA, W2V10, OUR-APP. All experiments were performed with the Geo880 dataset. We used 10-fold cross-validation.

LSTM Encoder-decoder - Geo880 Dataset			
Hidden Dim.	Method	S. E.	Accuracy
100	RANDOM	14.77	30.68
	TF-IDF/PCA	13.64	31.89
	W2V10	14.77	28.40
	OUR-APP	11.36	38.63
200	RANDOM	11.36	36.36
	TF-IDF/PCA	10.23	38.67
	W2V10	11.36	32.95
	OUR-APP	07.95	54.55
400	RANDOM	09.09	51.14
	TF-IDF/PCA	07.95	54.55
	W2V10	10.23	42.04
	OUR-APP	**06.81**	**64.77**

Table 5.1: *Implementation without neural attention*

[1] http://protege.stanford.edu/

[2] Another option would be to use a grammar that describes SPARQL, then use it as parameter of the parser algorithm. However a rejection by the SPARQL interpreter is a cheaper option found.

LSTM Encoder-Decoder - Geo880 Dataset			
Hidden Dim.	Method	S. E.	Accuracy
100	RANDOM	12.50	37.50
	TF-IDF/PCA	11.36	40.91
	W2V10	12.50	35.23
	OUR-APP	0	54.55
200	RANDOM	11.36	47.73
	TF-IDF/PCA	10.22	51.14
	W2V10	11.36	45.45
	OUR-APP	09.09	67.04
400	RANDOM	07.95	62.50
	TF-IDF/PCA	06.81	64.77
	W2V10	09.09	60.23
	OUR-APP	**05.68**	**78.40**

Table 5.2: *Implementation with neural attention*

In Table 5.2 the tests were run on the LSTM encoder-decoder architecture with attention mechanism. In next table we show examples of inputs and outputs generated by our model.

Input	Output (Without IRI Prefix to simplify)
how many rivers are there in idaho?	SELECT (COUNT(?A) AS ?QTD) { ?idaho p:river ?A FILTER (regex(str(?idaho), "idaho", "i")) . }
show major cities in colorado ?	SELECT ?A { ?colorado p:city ?A FILTER (regex(str(?colorado), "colorado", "i")) . ?A r:type p:Major . }
what are the cities of the state with the highest point ?	SELECT ?B { ?A p:city ?B . { SELECT ?A { ?A r:type p:State . ?A p:highest_point ?B . ?B p:height ?height . } ORDER BY DESC(?height) LIMIT 1 } }

Regarding the task of transforming the natural language into SPARQL, we compared our work with AlAgha (2015) and Kaufmann et al. (2006). In the first paper, the authors also use the Geo880 dataset and through linguistic analysis identify elements of natural language and generate RDF triples. In the second paper, the main strategy of the authors was to try to associate triples of natural language with RDF triples. As can be seen in the Table 5.6, we obtained better results in the tests with dataset Geo880.

	Accuracy
AlAgha (2015)	58.61
Querix Kaufmann et al. (2006)	77.67
Our method	78.40

Table 5.3: *Natural Language to SPARQL comparisons*

Although we mention the work Kaufmann et al. (2006), that also makes use of the Geo880 dataset, we do not make the comparison with it because it does not use the original set of Geo880 queries in their tests. With respect to the **syntactical errors**, we can see in our tests that the better the model in general, the lower the error rate of syntax. We the next section we present results on model that the generated sentence has syntactic guarantee.

5.4 Results on Encoder-CFG Decoder

In the experiments, we used 10-fold cross-validation. In this Section, we ran the experiments using a LSTM Encoder-CFG Decoder neural architecture with attention. In Table 5.4, we make a

comparison using different χ context sizes (see Algorithm 2).

context size	geoquery	jobs640	train time
3	76.53	78.36	4h
5	79.82	81.92	6h
∞	83.25	85.71	11h

Table 5.4: *Encoder-CFG Decoder with different context sizes*

We can see, by Table 5.4, that the best results were obtained considering $|\chi| = +\infty$. We use this result to make a comparison with our implementation of the known LSTM Encoder-Decoder models Bahdanau et al. (2014) used in Dong and Lapata (2016).

	Geoquery		Jobs640	
	Accuracy	S.E.	Accuracy	S.E.
LSTM Encoder-Decoder	78.40	5.68	80.02	4.79
LSTM Encoder-Decoder with Attention	82.31	3.84	84.62	2.94
LSTM Encoder CFG-Decoder	83.25	0	85.71	0
LSTM Encoder CFG-Decoder with Attention	85.02	0	87.35	0

Table 5.5: *Neural semantic parsing models comparisons*

Regarding the task of transforming the natural language into SPARQL, we compared our work with AlAgha (2015) and Kaufmann et al. (2006). In the first paper, the authors also use the Geo880 dataset and through linguistic analysis identify elements of natural language and generate RDF triples. In the AlAgha (2015), the main strategy of the authors was to try to associate triples of natural language with RDF triples. As can be seen in Table 5.6, we obtained better results in the tests with dataset Geoquery.

	Geoquery	Jobs640
AlAgha (2015)	58.61	-
Querix Kaufmann et al. (2006)	77.67	-
LSTM Encoder-Decoder with Attention	82.31	84.62
LSTM Encoder CFG-Decoder with Attention	**85.02**	**87.35**

Table 5.6: *Natural Language to SPARQL comparisons*

It is worth mentioning that in Dong and Lapata (2016), Tang and Mooney (2001), Zettlemoyer and Collins (2005), a kind of logical form of prolog was used as target language, whereas in our work we consider SPARQL as a target language. From the machine learning point of view, the complexity of the target language is relevant. For instance, in Geo880 dataset, the pair sentence-logical form $\{\langle x, y\rangle\}$ is

```
x: parse([what,is,the,density,of,texas,?])
y: answer(A,(density(B,A),const(B,stateid(texas))))
```

On the other hand, in our SPARQL version we have the pairs

```
x: what is the density of texas ?
y: SELECT((?pop/?area) AS ?density) {?texas p:pop ?pop.?texas p:area ?area}
```

Besides the complexity, built-in functions such as `density` also simplifies the Geo880 format. These differences give spaces for discussions and questions around comparisons between the approaches.

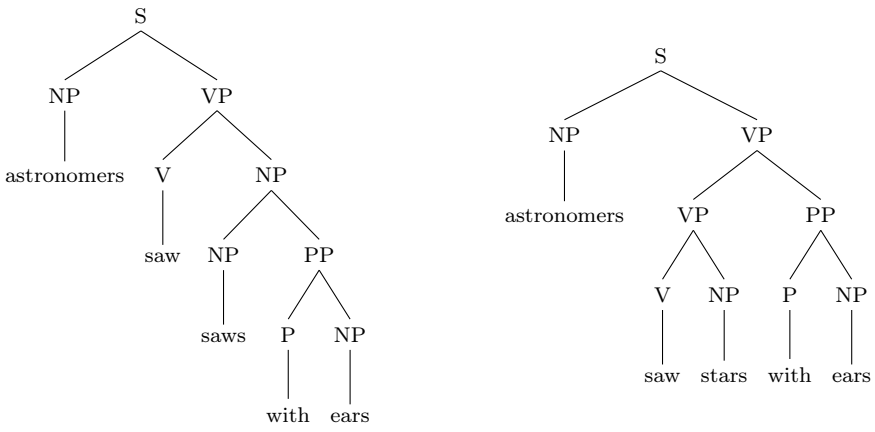
5.5 Discussions

Grammar The grammar structure we use in our work is a significant simplification of the original SPARQL's grammar [3]. We have made this simplification primarily because the original grammar version is not purely context-free. Another reason is the fact that, empirically, we have found that simpler grammars adapted to evaluation *corpora*'s domains achieve better results. The resulting simplification include the following characteristics:

- Unambiguous grammar: A context free grammar is called ambiguous if there exists more than one leftmost derivation or more than one rightmost derivation for a string which is generated by grammar (Sipser, 2006). For instance, given this grammar:

S	→	NP VP
PP	→	P NP
VP	→	V NP
VP	→	VP PP
NP	→	NP PP
P	→	with
V	→	saw
NP	→	astronomers
NP	→	ears
NP	→	saw
NP	→	stars

  There are two possible parse trees for the phrase: "astronomers saw stars with ears"



  Ambiguity is not a good feature for our model. This example was extracted from (Manning and Schütze, 1999).

- Non-recursive rules: recursive rules are nonterminal rules to which some element of the tail is equal of head element, this means that the rule can call itself. For example, in our previous grammar, **NP -> NP PP** and **VP -> VP PP** are recursive rules. In practice, lack of recursion implies that grammar is finite. A recursive CFG can have a compact and finite representation through non-recursive rules (Nederhof and Satta, 2002).

These characteristics impoverish the power of CFG, so we have a trade-off between grammatical power and learning ability. However, our first goal is to ensure syntacticity, regardless of grammar power. As a future work we intend to investigate grammars that work best for our model.

[3] https://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217/#grammar

Time performance It is important to stress that we have not focused on time performance. Instead, our main contributions focus on accuracy. This becomes clearer when we look closer to our main two contributions: Section 4.1 and 4.2. The first contribution consists in improving the accuracy of the training process. For this, we have an initial extra step in order to generate an attention matrix and thus the vector representation. From this step, the new vectorial representation is already established and can be used as many times as necessary.

In the second contribution, the number of neural networks to be trained increases according to the number of distinct non-terminals in the grammar. In addition, in each network interaction, the backpropagation step is greater because we also adjust the rules-related networks that appear later.

Chapter 6

Conclusions and Future Works

The purpose of this work was to explore artificial neural network resources in the development of a model that may be able to translate from natural language to SPARQL. The choice of the OWL Ontology and the SPARQL language as the target language is due to the fact that we are interested in practical applications. Among the advantages of using artificial neural networks, we can highlight the fact that we do not need linguistic knowledge nor do we depend on the development of complex grammars.

In addition to dealing with SPARQL, we propose in this work a representation of the target language lexicon that according to the results was a good approach. This representation is only possible because we use the concept of Matching of terms oriented by the alignment table that is generated by the mechanism of neural attention.

In the second contribution we present an extension of the LSTM Encoder-Decoder method that guarantees that the target language is in agreement with a given CFG. The results also show that the method we propose is competitive with others from the literature, therefore we give a relevant contribution to natural language translation.

The main points we observed during the development of this work and the execution of the experiments lead us to some conclusions: The model learning is more effective, structured and easy in the same proportion that the SPARQL corpus is structured and clear. It means that filters and other complex clauses must be added a posteriori; Although we did not put any restriction on the CFG, more complex ones may affect the training performance; The proposed training model is at least one and a half times slower than the traditional encoder-decoder model.

The form of translation developed in this work is very important for many other tasks. The fact of obtaining robustness in the target language syntax gives us more confiability to the translation problem where one does not care about the source grammar, but the target one.

Among the main future works, we can highlight: Explore more efficient means of representing the context; Tests with different versions of CFG capable of generating our training language; Tests with more modern datasets used in semantic parsing like ATIS (Hemphill et al., 1990) and WebQuestions (Berant et al., 2013).

Appendix A

Artificial Neural Network

A.0.1 From Perceptron to Recurrent Neural Networks

Early research on Neural Networks (NNs) began by (McCulloch and Pitts, 1943). In this work, they established the foundations of neurocomputation, developing the mathematical procedures that describe the functioning of biological neurons. This contribution was strictly conceptual, since the authors did not suggest practical applications for their work, nor the proposed systems had the ability to learn.

NNs were developed in an attempt to produce a computer model that simulates the structure and functioning of the human brain. A NN is a system that has computational power acquired by learning and generalization. The learning is related to the ability of NNs to adapt their parameters as a consequence of the interaction with the external environment. Generalization, in turn, is associated with the ability of these networks to provide consistent responses to data not shown during the training step.

NNs have a very simple structure of processing elements, inspired in the operation of the biological neuron (see Figures A.1, A.2), with connections between these processing elements. Each connection on the network has an associated weight, and this weight is the intensity of interaction or coupling between the processing elements and their nature is excitatory or inhibitory (Ellacott et al., 2012).

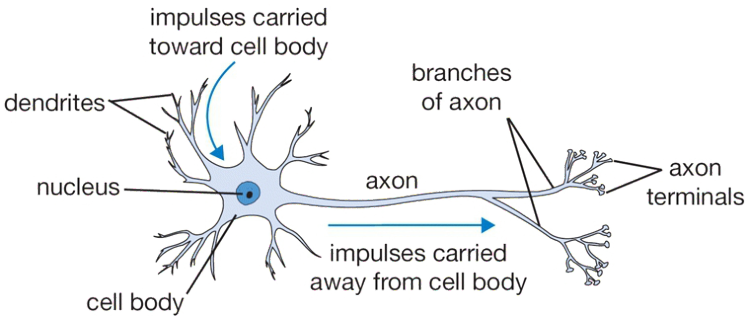


Figure A.1: *Biological neuron*

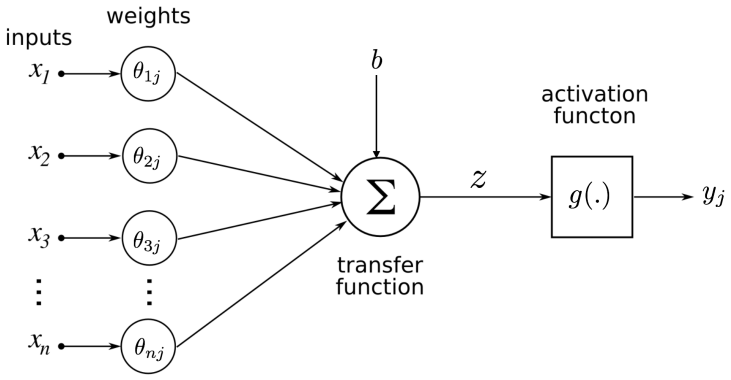


Figure A.2: *Artificial neuron*

Simplistically we can say that the dendrites of the biological model are associated with the n input terminals $(x_1, x_2, ..., x_n)$ of the artificial model to emulate the behavior of synapses. The nucleus is associated with the transfer function and the axon is associated with activation of the network and the output function. Although there are some discrepancies between terminologies, in our work the network transfer function is related with the multiplication operation of the inputs by weights and the sum of them, as the following equation:

$$z = b + \sum_{i=1}^{n} x_i \theta_i. \tag{A.1}$$

Some weights have excitatory signals (+) and other inhibitory signals (-). The input values and activation of neurons may be discrete (in general the values are 0 and 1 or -1 , 0 and 1), or continuous, generally comprised in the ranges [0,1] or [-1,1]. We also need a rule which gives the effect of the total input on the activation of the unit, this is called activation function (Kröse et al., 1993). In general, the activation function is a nonlinear function (i.e sigmoid function, hyperbolic tangent function, logistic function).

In a more practical way, we can explain a neural network in terms of a generalization of a logistic function (Pampel, 2000) if the network activation function is a logistic function. Let's summarize the main points of a logistic regression to better visualize the relationship between the two concepts.

In a logistic regression, initially we are looking for a function that returns the probability of a certain fact given their explanatory variables. In this case, we call explanatory variables the characteristics of this fact. For instance, we want to classify cancer tumors as either malignant or non-malignant, where the variables could be the size of the tumor or the presence of some genome. This function could be called h_θ, which represents our hypothesis:

$$0 \leq h_\theta(x) \leq 1.$$

These explanatory variables make up the linear Function $\theta^T \mathbf{x}$ where the vector $\mathbf{x}$ is the set of explanatory variables and the vector θ is a vector of the weights that will be learned.

In general, the ratio between the explanatory variables and probability is not a linear relationship. This is due to several reasons, among them, the most obvious is that probabilities are limited between the values 0 and 1 and linear functions are not.

The logistic function:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{A.2}$$

where z is a linear function, represents very well the relation between probabilities and explanatory variables.

We can define h_θ function as a follow:

$$h_\theta(x) = g(\theta^T \mathbf{x}).$$

Given a set of training with pairs $\{(\hat{x}^{(1)}, \hat{y}^{(1)}), ..., (\hat{x}^{(m)}, \hat{y}^{(m)})\}$, the cost function fits θ parameters to get the minimum error (less cost) in the training process. The error value is obtained from the difference between $\hat{y}$ and $h_\theta(\hat{x}) = y$. Where $\hat{y}$ means that is the output value that is in the training set, and y is the value obtained by hypothesis.

The cost function of logistic regression:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{cost}(h_\theta(x^i), y^i), \tag{A.3}$$

is easy to be understood, differently from the linear regression cost function, here we used a nonlinear function (*cost*). For each value $\hat{y}$ from training data, we use a different function to get error. The *cost function* can be defined:

$$\text{cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_{\theta(x)}) & \text{if } y = 0 \end{cases} \tag{A.4}$$

Then, if the $h_\theta(\hat{x})$ tends to 1, for values close to 1 it is a value close to zero (in this case, the parameter is good). Otherwise, if $h_\theta \hat{x}$ tends toward 0, $-\log(y)$ tends toward very large numbers thereby increasing the value of the error.

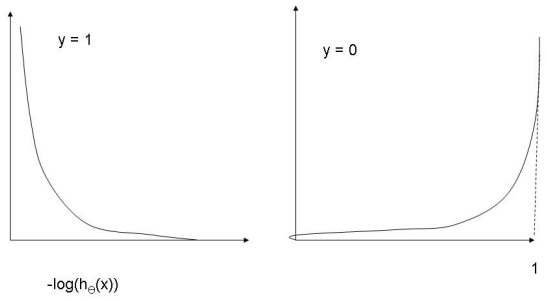


Figure A.3: *Cost cases*

We can summarize the logistic regression cost function:

$$J(\theta) = \frac{1}{m} \left[\sum_{i=1}^{m} -y^i \log(h_\theta(x^i)) - (1 - y^i) \log(1 - h_\theta(x^i)) \right] + R(\theta) \tag{A.5}$$

where $R(\theta)$ is a regularization term that is generally used like a penalty to prevent unwanted learning (for example, overfitting).

After this brief summary of logistic regression, we will return to the neural networks to compare both architectures. According to Figure A.2 we can extract the following equation (perceptron):

$$g\left(b + \sum_{i=1}^{n} x_i \theta_i \right) \tag{A.6}$$

Each neuron is equivalent to a logistic regression architecture. Thus, the cost function of the

neural network can so be defined:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^i \log(h_\theta(x^i)) + (1 - y_k^i)\log(1 - (h_\theta(x^i))_k)\right] + R(\theta)$$

In fact, a neuron can be compared to a logistics unit. However, neural networks can be very powerful, formed by many neurons and other types of non-linear functions.

Regularization term

Regularization, in neural networks, is process of introducing additional information in order to solve an ill-posed problem[1] or to prevent overfitting (Duda et al., 2012). This information works as a penalty for the methods and is generally added in the cost function. Regularization terms can be motivated as a technique to improve the generalization of a learned model.

In a more simple way, you can see the mathematical intuition behind a regularization term for a linear regression. For example, in Figure A.4 we have a solution to the classification between balls and crosses.

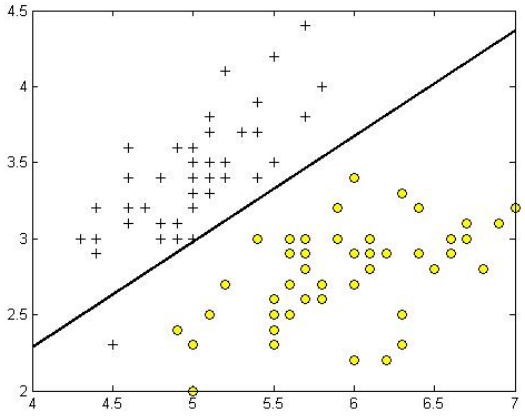


Figure A.4: *One solution to classifier problem*

Although the solution shown in Figure A.4 is valid, we can see clearly that it is a biased solution because the line that cuts the two classes is much closer to the classes of crosses. In this case, we can apply a *linear regularization term* $\lambda||\theta||_2^2$ necessarily leading to a greater balance between the values of the weights. Similarly we can also apply the regularization term for logistic regression. In logistic regression we also have problems with very large θ and we can apply the following regularization term:

$$\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

where λ is a constant, m is the number of the training examples and n is the quantity of θ. Similarly, we can apply a regularization term in neural networks.

$$\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{sl}\sum_{j=1}^{sl+1}(\theta_{ji}^{(l)})^2$$

In the neural network we need to consider the amount of layers L, where each l is a specific layer and sl is the number of neurons in each layer. The index i and j refer to the connections between

[1] In this case, we can consider a ill-posed problem that which have a small set of training turning difficult a generalization. The regularization term alleviates a separator function that is bad

neurons of one layer to the next layer's.

Multilayer perceptron

A neural network formed by a single artificial neuron, as shown in Figure A.2, is limited to solving linearly separable problems. Multilayer Perceptron (MLP) is a modification of the standard linear perceptron and it can distinguish data that is not linearly separable. A MLP is a model of a feedforward neural network that maps input data sets for a set of desired outputs. A MLP consists of several layers of nodes in a directed graph, with each fully connected to the next layer. Except for the input nodes, each node is a neuron with a non-linear activation function. MLP uses a supervised learning technique called backpropagation to train the network (Rumelhart et al., 1985).

There are various parameters that are part of the definition of network architecture, such as the number of network layers, the number of neurons in each layer, the type of connection and the network topology.

Training a neural network

A neural network has to be configured such that the application of a set of inputs produces (either "directly" or via a relaxation process) the desired set of outputs. Various methods to set the strengths of the connections exist. One way is to set the weights explicitly, using a priori knowledge. Another way is to "train" the neural network by feeding it with teaching patterns and letting it change its weights according to some learning rule (Kröse et al., 1993).

Paradigms of learning: We can categorize the learning processes in two distinct sets. These are:

- Supervised (or Associative) learning in which the network is trained by providing it an input and matching output patterns.These input-output pairs can be provided by an external teacher, or by the system which contains the network (self-supervised).
- Unsupervised learning or Self-organization in which an output unit is trained to respond to clusters of pattern within the input. In this paradigm the system is supposed to discover statistically salient features for the input population. Unlike the supervised learning paradigm, there is not a priori set of categories into which the patterns are to be classified; rather the system must develop its own representation of the input stimuli.

Gradient descent: The idea of using the derivative of the error function to transform it in learning procedure comes from the optimization. In training you want to minimize the cost function (see equation A.7) its implies finding values for θ such that the value of cost function (error) is minimized, given a cost function parameterized by θ:

$$\min_{\theta} J(\theta). \tag{A.7}$$

The gradient descent method uses the derivative of the cost function to walk in the opposite inclination of the derivative, subtraction operation of theta value by the derivative of the cost function with respect to θ, in an attempt to find θ parameters that minimize the cost function. If the cost function is a convex function (like Figure A.5) then the method of descent by the gradient usually find the optimal solution (global minimum).

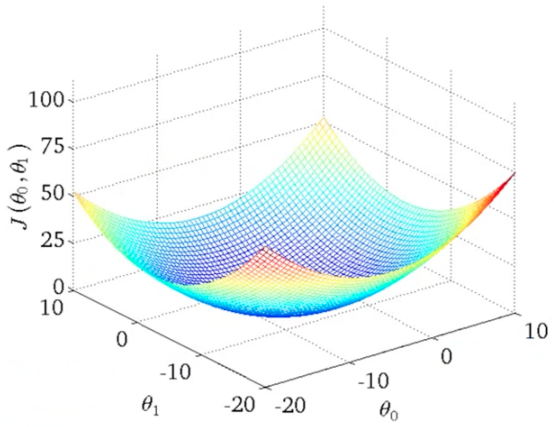


Figure A.5: *Convex cost function*

Usually, only neural networks without intermediate layers may have a convex cost function. In most of them, the cost functions of a multi-layer neural network typically have many valleys and hills resembling Figure A.6.

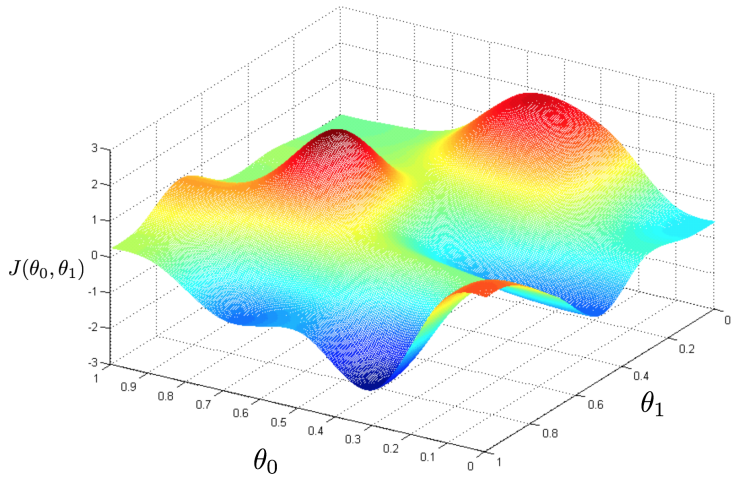


Figure A.6: *Nonconvex cost function*

The following is a simple version of the gradient descent. The α term is known as the error rate and it defines how much adjustment we will have in each θ_j, where $0 \leq j \leq n$ and n the quantity of θ parameters.

function GRADIENT-DESCENT(a)
  repeat
    $\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$

  until convergence
  return θ
end function

While we apply the gradient descent we progressively find θ values that minimize the J function.

Backpropagation: an abbreviation for "backward propagation of errors", it is a common method of training artificial neural networks used in conjunction with an optimization method such as gradient descent. The method calculates the gradient of a loss function respecting all the weights in the network. The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function.

Backpropagation requires a known, desired output for each input in order to calculate the loss function gradient. It is therefore usually considered to be a supervised learning method, although it is also used in some unsupervised networks such as autoencoders. It is a generalization of the delta rule to multi-layered feedforward networks, made possible by using the chain rule to iteratively compute gradients for each layer. Backpropagation requires that the activation function used by the artificial neurons (or "nodes") to be differentiable.

Now let's take an example of how to calculate the backpropagation error to a multilayer neural network. We use a network of 4-layers with the settings shown in Figure A.7.

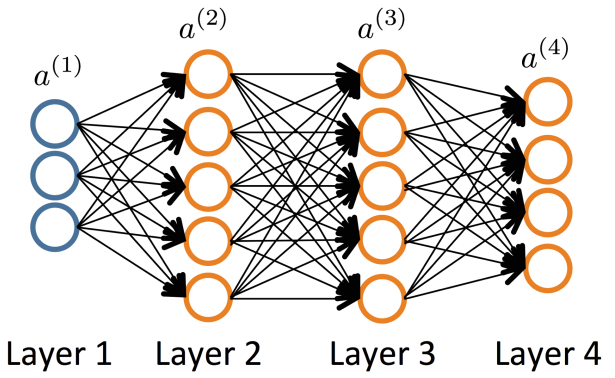


Figure A.7: *Neural network with 4 layer*

Before calculating the backpropagation error for this network we will create intermediate variables that will help us to simplify the problem. The function g is a nonlinear function which in our case is the logistic function, and $a^{(l)}$ is the variable created to capture the outputs of each layer of the network. The variable z receive the output of $a^{(l)}$ and multiplied it by their respective weights; a_0 is the bias.

$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)})(\text{add } a_0^{(2)}) \\
z^{(3)} &= \theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)})(\text{add } a_0^{(3)}) \\
z^{(4)} &= \theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\theta(x) = g(z^{(4)})
\end{aligned}
$$

The **add** function is a vector sum operation. The next step is to calculate the δ's for each unit of the network. Intuitively, we can interpret the variable $\delta_j^{(l)}$ as being the error of j node in the layer l. For each unit of the last layer (i.e. $L = 4$), we can define delta as:

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

From the penultimate layer back to the second one, we use a different method to calculate the deltas:

$$
\begin{aligned}
\delta^{(3)} &= (\theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)}) \\
\delta^{(2)} &= (\theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})
\end{aligned}
$$

The $\delta^{(1)}$ stays out because it refers to the data input layer which has no error associated with it. The function g' is a derivate from g. For example, take this function: $g'(z^{(3)})$ is equal to $a^{(3)} .* (1 - a^{(3)})$. The signal (.*) represents element-by-element product. We can calculate δ from

the following equation:

$$\delta_j = \frac{\partial J(\theta)}{\partial a_j}\frac{\partial a_j}{\partial z_j} = \begin{cases} (a_j - y_j)a_j(1-a_j) & \text{if } j \text{ is output neuron} \\ (\sum_{l \in L} \delta_l \theta_{jl})a_j(1-a_j) & \text{if } j \text{ is inner neuron} \end{cases}$$

In a more general way, we can use the previous functions to develop an error calculation algorithm.

Input: training set $\{(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})\}$.
Initialization:
Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j)

- For $i = 1$ to m
  - Set $a^{(1)} = x^i$
  - Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, ..., L$
  - Using $y^{(i)}$ compute $\delta^{(L)} = a^{(L)} - y^{(i)}$
  - Compute $\delta^{(L-1)}, \delta^{(L-2)}, ..., \delta^{(2)}$
  - $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)}\delta_i^{(l)}$

$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} + \theta_{ij}^{(l)}$ if $j \neq 0$
$D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)}$ if $j = 0$

Figure A.8: *The Backpropagation algorithm*

The variable Δ refers to the set of δ's and the variable D:

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = D_{ij}^{(l)}$$

Bibliography

Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, 2003. 29

Iyad AlAgha. Using linguistic analysis to translate arabic natural language queries to sparql. *arXiv preprint arXiv:1508.01447*, 2015. 2, 21, 37, 38

Ali Mohamed Nabil Allam and Mohamed Hassan Haggag. The question answering systems: A survey. *International Journal of Research and Reviews in Information Sciences (IJRRIS)*, 2(3), 2012. 21

I. Androutsopoulos, G.D. Ritchie, and P. Thanisch. Natural language interfaces to databases-an introduction. *arXiv preprint cmp-lg/9503016*, 1995. 2

Marcelo Arenas and Jorge Pérez. Querying semantic web data with sparql. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 305–316. ACM, 2011. 1

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. *The semantic web*, pages 722–735, 2007. 21, 27

Emmon Bach. Control in montague grammar. *Linguistic inquiry*, 10(4):515–531, 1979. 1

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. xiii, 1, 8, 14, 15, 16, 17, 22, 38

S. Bechhofer, F. Van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein, et al. Owl web ontology language reference. *W3C recommendation*, 10:2006–01, 2004. 17

D. Beckett. Sparql rdf query language reference v1. 8. *Consultado (08/02/2012) en [http://www. dajobe. org/2005/04-sparql/]*, 2006. 19

Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. Neural probabilistic language models. *Journal of Machine Learning Research*, 3:1137–1155, 2003. 10, 22, 27

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, 2013. 41

Chris Biemann. Ontology learning from text: A survey of methods. In *LDV forum*, volume 20, pages 75–93, 2005. 3

Christopher M Bishop. Pattern recognition. *Machine Learning*, 2006. 9

Christian Bizer and Andreas Schultz. The berlin sparql benchmark. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(2):1–24, 2009. 1

Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee. Linked data on the web (ldow2008). In *Proceedings of the 17th international conference on World Wide Web*, pages 1265–1266. ACM, 2008. 21

W.N. Borst. *Construction of engineering ontologies for knowledge sharing and reuse*. Universiteit Twente, 1997. 17

Carlos Buil-Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. Sparql web-querying infrastructure: Ready for action? In *International Semantic Web Conference*, pages 277–293. Springer, 2013. 1

Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering sparql queries over relational databases. *Semantic Web*, 8(3):471–487, 2017. 1

Yonggang Cao, Feifan Liu, Pippa Simpson, Lamont Antieau, Andrew Bennett, James J Cimino, John Ely, and Hong Yu. Askhermes: An online question answering system for complex clinical questions. *Journal of biomedical informatics*, 44(2):277–288, 2011. 3

Leandro Menezes Capetta. Consulta a banco de dados em linguagem natural. *Revista OMNIA Exatas*, 4(1):72–80, 2012. 2

Yung-Chun Chang, Yu-Lun Hsieh, Cen-Chieh Chen, and Wen-Lian Hsu. A semantic frame-based intelligent agent for topic detection. *Soft Computing*, pages 1–11, 2015. 3

Jean-Cédric Chappelier, Martin Rajman, et al. A generalized cyk algorithm for parsing stochastic cfg. *TAPD*, 98(133-137):5, 1998. 20

Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. *AAAI/IAAI*, 2005(598-603):18, 1997. 20

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. *Syntax, Semantics and Structure in Statistical Translation*, page 103, 2014a. 1, 14, 22

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014b. xiii, 1, 13, 14, 22

Jan Chorowski, Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. End-to-end continuous speech recognition using attention-based recurrent nn: first results. *arXiv preprint arXiv:1412.1602*, 2014. 14

Maurizio Corbetta and Gordon L Shulman. Control of goal-directed and stimulus-driven attention in the brain. *Nature reviews neuroscience*, 3(3):201–215, 2002. 16

Colin De la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010. 20

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990. 11

Li Dong and Mirella Lapata. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*, 2016. 1, 3, 4, 22, 23, 24, 34, 35, 38

Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012. 46

Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970. 20, 32

Stephen W Ellacott, John C Mason, and Iain J Anderson. *Mathematics of neural networks: models, algorithms and applications*, volume 8. Springer Science & Business Media, 2012. 43

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990. 10

Sébastien Ferré. Squall: A controlled natural language for querying and updating rdf graphs. In *International Workshop on Controlled Natural Language*, pages 11–25. Springer, 2012. 21

Sébastien Ferré. Squall: The expressiveness of sparql 1.1 made available as a controlled natural language. *Data & Knowledge Engineering*, 94:163–188, 2014. 3

Ruifang Ge and Raymond J Mooney. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the ninth conference on computational natural language learning*, pages 9–16. Association for Computational Linguistics, 2005. 1

Omer Goldman, Veronica Latcinnik, Udi Naveh, Amir Globerson, and Jonathan Berant. Weakly-supervised semantic parsing with abstract examples. *arXiv preprint arXiv:1711.05240*, 2017. 22

Alan Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE, 2013. xiii, 8, 16

Alex Graves. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*, 2012. 1

T.R. Gruber et al. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993. 17

David Guthrie, Ben Allison, Wei Liu, Louise Guthrie, and Yorick Wilks. A closer look at skip-gram modelling. In *LREC*, pages 1222–1225, 2006. 11

Zellig Harris. Mathematical structures of language. *Interscience tracts in pure and applied mathematics*, 1968. 11

Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. Executing sparql queries over the web of linked data. In *International Semantic Web Conference*, pages 293–309. Springer, 2009. 1

Charles T Hemphill, John J Godfrey, and George R Doddington. The atis spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*, 1990. 41

Pascal Hitzler, Markus Krotzsch, and Sebastian Rudolph. *Foundations of semantic web technologies*. Chapman and Hall/CRC, 2009. 1

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Master's thesis, Institut fur Informatik, Technische Universitat, Munchen*, 1991. 5

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997. 5, 6, 12

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001. 5

M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe. A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools edition 1.0. *University of Manchester*, 2004. xiii, 18

Robin Jia and Percy Liang. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 12–22, 2016. 22, 24

Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002. 29

Boris Katz. Annotating the world wide web using natural language. In *RIAO*, pages 136–159, 1997. 3

E. Kaufmann, A. Bernstein, and R. Zumstein. Querix: A natural language interface to query ontologies based on clarification dialogs. In *5th International Semantic Web Conference (ISWC 2006)*, pages 980–981, 2006. 21, 37, 38

G. Klyne, J.J. Carroll, and B. McBride. Resource description framework (rdf): Concepts and abstract syntax. *W3C recommendation*, 10, 2004. 19

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1516–1526, 2017. 22, 24, 25

Ben Kröse, Ben Krose, Patrick van der Smagt, and Patrick Smagt. An introduction to neural networks. 1993. 44, 47

Nate Kushman and Regina Barzilay. Using semantic unification to generate regular expressions from natural language. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 826–836, 2013. 24

Stanislao Lauria, Guido Bugmann, Theocharis Kyriacou, and Ewan Klein. Mobile robot programming using natural language. *Robotics and Autonomous Systems*, 38(3):171–181, 2002. 3

Nicholas Locascio, Karthik Narasimhan, Eduardo De Leon, Nate Kushman, and Regina Barzilay. Neural generation of regular expressions from natural language with minimal domain knowledge. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1918–1923, 2016. 22, 24

Kevin Lund and Curt Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208, 1996. 11

Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015. 17, 22

Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, page 89. Presses universitaires de Louvain, 2015. 4

Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999. 39

Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pages 403–415. Springer, 2013. 3

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. 43

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010. 29

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a. 11

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b. xiii, 11, 12

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013c. 11

Jeff Mitchell and Mirella Lapata. Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429, 2010. 11, 12

Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, pages 2204–2212, 2014. 14

Raymond J Mooney. Learning for semantic parsing. In *Computational Linguistics and Intelligent Text Processing*, pages 311–324. Springer, 2007. 3

Mark-Jan Nederhof and Giorgio Satta. Parsing non-recursive context-free grammars. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 112–119. Association for Computational Linguistics, 2002. 39

Fred C. Pampel. *Logistic Regression: A prime*. Sage Publication, 2000. 44

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014. 27, 28

Fernando CN Pereira and David HD Warren. Definite clause grammars for language analysis?a survey of the formalism and a comparison with augmented transition networks. *Artificial intelligence*, 13(3):231–278, 1980. 24

E. Prud, A. Seaborne, et al. Sparql query language for rdf. 2006. 18, 19

R.J. Rabelo. Ontologia em sistemas de agentes. 2011. 17

Robert Rodman. The proper treatment of relative clauses in a montague grammar. *Papers in Montague Grammar*, (2):80, 1972. 1

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985. 47

Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *arXiv preprint arXiv:1509.00685*, 2015. xiii, 17

Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975. 11

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681, 1997. 8, 16

Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006. 20, 39

Tommaso Soru, Edgard Marx, Diego Moussallem, Gustavo Publio, André Valdestilhas, Diego Esteves, and Ciro Baron Neto. Sparql as a foreign language. *arXiv preprint arXiv:1708.07624*, 2017. xiii, 22

George J Suci and Percy H Tannenbaum. The measurement of meaning. *Urbana: University of Illinois Press*, 1957. 10, 11

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014. 1

V. Tablan, D. Damljanovic, and K. Bontcheva. A natural language query interface to structured information. *The Semantic Web: Research and Applications*, pages 361–375, 2008. 2

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. 2015. 4

Lappoon R Tang and Raymond J Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Machine Learning: ECML 2001*, pages 466–477. Springer, 2001. 2, 35, 38

Cynthia Thompson. Acquiring word-meaning mappings for natural language interfaces. *Journal of Artificial Intelligence Research*, pages 1–44, 2003. 2

Mikolov Tomáš. *Statistical language models based on neural networks*. PhD thesis, PhD thesis, Brno University of Technology. 2012.[PDF], 2012. 10, 27

Yushi Wang, Jonathan Berant, and Percy Liang. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1332–1342, 2015. 23, 24

David HD Warren and Fernando CN Pereira. An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*, 8(3-4):110–122, 1982. 2

Terry Winograd. Understanding natural language. *Cognitive psychology*, 3(1):1–191, 1972. 1

Chunyang Xiao, Marc Dymetman, and Claire Gardent. Sequence-based structured prediction for semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1341–1350, 2016. xiii, 23, 24

Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015. 14, 17

John M Zelle and Raymond J Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1050–1055, 1996. 2

Luke S Zettlemoyer and Michael Collins. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 658–666. AUAI Press, 2005. 2, 4, 35, 38