Czech Technical University in Prague
Faculty of Electrical Engineering

Relational Learning with Neural Networks for Machine Translation

Master Thesis

Daniel Krynský

Study Programme: Open Informatics
Branch of Study: Artificial Intelligence

Advisor: Ing. Gustav Šourek

Prague, May, 2018

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science

DIPLOMA THESIS AGREEMENT

Student: Krynský Daniel

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: Relational Learning with Neural Networks for Machine Translation Evaluation

Guidelines:

Machine translation is a complex discipline exhibiting many features of both statistical and symbolic reasoning. Currently, the field is dominated by neural networks, bulding on latent, vector representations, ignoring the relational structure of sentences. The goal of this work is to propose and implement a relational machine learning method that would leverage both structure and statistical properties of translated sentences aimed at better, more human-like, performance evaluation in machine translation.

1) Get familiar with natural language processing methods for machine translation.
2) Review evaluation metrics used in machine translation and assess their pros and cons. Focus on human evaluation (e.g. HUME) methods.
3) Get familiar with Statistical Relational Learning, particularly the framework of Lifted Relational Neural Networks.
4) Propose a strategy to apply a chosen framework for learning of a proper machine translation evaluation metric(s).
5) Benchmark your approach on existing structured corpora (e.g. UCCA) for machine translation.

Bibliography/Sources:

[1] Alexandra Birch, et al. : 'HUME: Human UCCA-Based Evaluation of Machine Translation.' arXiv preprint arXiv:1607.00030 (2016).
[2] Philipp Koehn : 'Statistical Machine Translation.' Cambridge University Press New York, 2010.
[3] Gustav Šourek, Vojtěch Aschenbrenner, Filip Železný and Ondřej Kuželka : 'Lifted Relational Neural Networks.' NIPS CoCo 2015: Cognitive Computation: Integrating Neural and Symbolic Approaches. 2015.

Diploma Thesis Supervisor: Ing. Gustav Šourek

Valid until the end of the winter semester of academic year 2018/2019

prof. Dr. Michal Pěchouček, MSc.
Head of Department

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická -7-
L.S.

prof. Ing. Pavel Ripka,CSc.
Dean

Prague, July 21, 2017

Acknowledgments

I would like to thank my advisor Gustav Šourek for the immense amounts of help and support. My thanks also go to Ondřej Bojar for providing me with the data needed for the practical part of this thesis.

Declaration

I declare, that I have created this thesis on my own and that I have quoted every information source used. This has been done according to the methodical directive about adhering to ethical principles during preparation of university final projects.

in Prague, 21. May, 2018

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

v Praze, 21. května, 2018

Daniel Krynský

Abstract

The goal of this thesis is to evaluate the use of neural networks, which are based on a relational approach for automatic evaluation of machine translation. This approach makes use of a set of relational rules, which serve as a template and which, together with relational data can be used for creating neural networks. For the machine translation quality evaluation a metric called Human UCCA-based Machine Translation Evaluation is used, both for leaves and internal nodes of tree representation of the input data. Part of the experiment will be comparison of the evaluation of translation from English to four different languages. Relational approach will be compared with feature-based approach with the use of neural networks.

Abstrakt

Tato práce má za cíl vyhodnotit použití neuronových sítí založených na relačním přístupu pro automatickou evaluaci strojového překladu. Tento přístup využívá sadu relačních pravidel, která slouží jako template a která, spolu s relačními daty, může být využita pro vytvoření neuronových sítí. Jako ohodnocení kvality strojového překladu je použita metrika Human UCCA-based Machine Translation Evaluation a to na listech i vnitřních uzlech stromové reprezentace vstupních dat. Součástí experimentu bude porovnání evaluace překladu z Angličtiny do čtyřech různých jazyků. Relační přístup bude porovnán s feature-based přístupem s využitím neuronových sítí.

Keywords

Artificial neural networks, Relational learning, Natural language processing, Predicate logic, Machine translation, Evaluation metrics

Klíčová slova

Umělé neuronové sítě, Relační učení, Zpracování přirozeného jazyka, Predikátová logika, Strojový překlad, Evaluační metriky

Contents

1. Introduction

The idea of machine translation of a text or a speech from one human language to another is not a new phenomenon. The attempts to have some sort of automated way of translation may be traced way back, even before computers were available. In its beginnings, machine translation had much to do with cryptography, because code breaking and tracking foreign ciphered messages was one of the main tasks for which the early computers were used. Later on the focus was more on the machine translation of technical, scientific as well as legislative documents. Hutchins (2005) gives a quick overview of this topic. Currently, machine translation is used for work with various types of online content. The ultimate goal of machine learning research in the field of machine translation is to have a system, which would produce translated text, that would be no worse than a translation of the same text by a human being. Ideally, this should work between any two languages. So far however, this doesn't exist. There is always some error on the output. In some cases, the error might be very low, in fact low enough, that the translation may be labeled as an acceptable translation. The main idea of this thesis is to use deep learning and to look at ways how the quality of machine translation could be evaluated automatically. Basically, if we have a source text and it's machine translation to another language, we want to be able to decide whether the translation is good enough to be acceptable by a set of introduced standards or not.

This thesis consists of three main parts. The first part describes fundamental theoretical principles behind Natural Language Processing (NLP). It explains what statistal NLP is and describes language mod-

els which are commonly used in NLP. Then it describes how sentences can be represented as tree structures and how additional syntactic, semantic and morphological information can be stored within these tree structures. Finally, the UCCA semantic representation scheme and the HUME semantic evaluation measure will be described. These two concepts will be used in the most significant part of this thesis, which will be learning using relational rules.

The second part of this thesis talks about a feature-based approach towards classification of machine translation with the use of neural networks. In this chapter we are going to define a feature vector and we are going to use existing tools and functions for creating and learning a neural network. The results obtained will serve as values for comparison with the main part of this thesis, to see whether we can get any better with relational-based approach.

The third and most important part of this thesis is the part dedicated to relational-based approach. We will work with annotated trees representing translations of a text from English to four target languages, Czech, German, Polish and Romanian, which will serve as our input data. Several relational rules sets will be created and run on several sets of input data. These input data sets will include separate sets for each target language as well as combination of two or more target languages to see whether learning on more languages combined can bring us better results. Testcases with rules sets taking into account semantics will also be performed.

In the chapter Experiments, results from the previous two parts will be presented, compared and discussed. Based on all the results we'll have, the best approach will be chosen as a starting point for possible future work.

2. Natural language processing

2.1 Statistical NLP

Statistical Natural Language Processing describes a process, whose input is a text written in a natural language and which outputs statistical data about this text. People have been interested in statistical data since always. One such example can be found in the field of cryptography called cryptoanalysis. The substitution cipher, which encodes a text by replacing the letters in the plaintext to obtain the ciphertext, can be broken by frequency analysis. The frequency of all letters are counted in the ciphertext, then the most frequent letters are replaced by the most common letters of the alphabet. In English, the most common letters are E, then T, then A. In a similar way, frequency analysis can be used to find the most frequent pairs or groups of letters. The basic statistical data we are interested in are typically frequencies of words, collocations and phrases. Individual words of collocations sometimes don't stand right next to each other. For such cases we may be interested in another statistical information, so called *mean offset*. Say, there is an article about people who

are drinking tea. The article contains following collocations:

a) cup of tea
b) cup of black tea
c) cup of strong tea
d) cup of very strong tea

We can compute the *mean offset* between *cup of* and *tea* as follows:

$\frac{1}{4}(1 + 2 + 2 + 3) = 2.0$

Other information which we are interested in are the deviations of the individual offsets from the mean. This is called variance and here is how to compute it:

$s^2 = \frac{\sum_{i=1}^{n}(d_i - d)^2}{n-1}$
n - how many times the collocation occur in the text
d_i - offset for the collocation i
d - mean offset

For our example with the cup of tea the variance is computed as:

$$s = \sqrt{\frac{1}{4-1}((1 - 2.0)^2 + (2 - 2.0)^2 + (2 - 2.0)^2 + (3 - 2.0)^2)} \approx 0.67$$

If the variance is low, then the distance between the two words is usually about the same. If the variance is zero, then the words within a collocation have always the same distance.

2.2 Language models

In NLP, language model is understood to be a probability distribution. Manning et. al. (2008) describes language model as a function, which puts a probability measure over strings, which are drawn from a certain vocabulary. There are situations where we are interested in how many and what words follow after some other words. Typ-

ing first few words of a phrase into a search engine will offer a user the most common words which follow in the current context. Other situation could be the study of verb valency, the capability of verbs to control (bind) other verb arguments. Let's consider a scenario where, reading some text, we obtained a sequence of words *W*.

$W = (w_1, w_2, ..., w_n)$

Based on the chain rule in probability theory we get the probability *p(W)*.

$p(W) = p(w_1, w_2, ..., w_n) = p(w_1)\,\mathrm{x}\,p(w_2|w_1)\,\mathrm{x}\,...\,\mathrm{x}\,p(w_n|w_1, w_2, ..., w_{n-1})$

Theoretically, we could compute the probability *p(W)* using all the words which have been read from the beginning until w_n. To make things less complex, we only work with a certain number of the most recently read words. Such a subsequence of words is called an *n-gram*. There are two reasons for this: working with too many variables isn't practical and we do not have an unlimited memory.

2.2.1 n-gram Language Model

n-gram is defined as a sequence of n items, typically words, in a given text. Terminology for *n-gram* names is based on the value of n:

n	**n-gram name**
1	unigram
2	bigram (or digram)
3	trigram

For the values of $n \geq 4$, *n-gram* names follow a pattern, where n is

replaced by the number of subsequent elements.

4 four-gram (or 4-gram)
5 five-gram (or 5-gram)
etc.

The value of n says how far back we look when parsing the text, i.e. with 4-gram model, we look at the current word plus the three preceding words.

2.2.2 n-th order Markov model

The task of language modeling is to correctly predict the next word given certain number of previous words (Manning C.D., Schütze H. (2010), p.191). The most efficient (in terms of memory efficiency) would be such a model, which would be able to predict the next word solely based on the current word. Markov model represents a general idea for this type of behaviour. In Markov model, we have a set of states and we are able to move from one state to another state with a certain probability, but there is no impact of any of the previously visited states on where to go from the state, we are currently at. Yes, such a language model would require no extra memory. Once we moved from one state to another by reading the next word on the input, all the history which has been read so far, could be thrown away and forgotten. This characteristic of a process would satisfy the definition of Markov property - the conditional probability distribution of future states depends only on the current state, not on any of the previous states (Serfozo R. (2009), p.2). Since we don't want to throw away the history immediately, we need to introduce more memory into our model by using a higher order Markov model. In order to compute the probability *p(W)*, we need to know (and remember) not only the current state (i.e. the current word), but

also n preceding states (words).
For an n-th order Markov model it holds that:
$p(x_i|x_{i-1}, x_{i-2}, ..., x_1) = p(x_i|x_{i-1}, ..., x_{i-n})$

Consider the example in Figure 2.1. There is a set of four states $S = \{x_1, x_2, x_3, x_4\}$ and we are able to get from one state to any other state with some probability. Once we visit all states, we'll automatically get to the end. The value of n is for example 1. So we have the 1st order Markov model, which means, that the probability of getting to a state depends on one previous state.

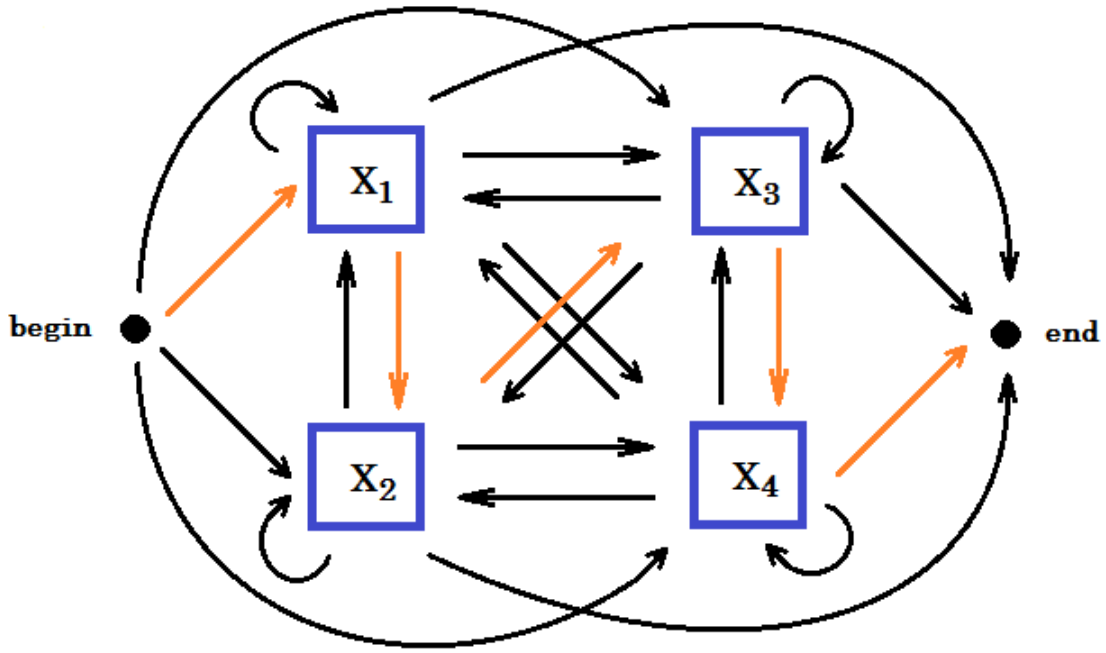


Figure 2.1: 1st order Markov model

We want to know the probability of going through all the states in this order: x_1, x_2, x_3, x_4. According to the general product rule (chain

rule) $p(x_1, x_2, x_3, x_4)$ would be calculated as:

$$p(x_1, x_2, x_3, x_4) = p(x_1) \text{ x } p(x_2|x_1) \text{ x } p(x_3|x_2, x_1) \text{ x } p(x_4|x_3, x_2, x_1)$$

but since we are working with the 1st order Markov model, we can simplify the formula as follows:

$$p(x_1, x_2, x_3, x_4) = p(x_1) \text{ x } p(x_2|x_1) \text{ x } p(x_3|x_2) \text{ x } p(x_4|x_3)$$

The general formula to compute the probability of the 1st order Markov model for k states is then:

$$p(x_1, x_2, ..., x_k) = p(x_1) \text{ x } \prod_{i=2}^{i=k} p(x_i|x_{i-1})$$

As we can see, the n-th order Markov model is a general idea of a process for which, in the field of Natural Language Processing, the n-gram Language Model is used.

2.2.3 Maximum Likelihood Estimation (MLE)

MLE is a method of estimating the values of parameters, given some observed data, by taking the values of parameters which maximize the likelihood of observed data. The task in MLE is to find a distribution function which, with the highest likelihood, represents the data.
In our case, we'll look at MLE example, which works with n-grams. We are interested in probabilities $p(w_1, ...w_n)$ and $p(w_n|w_1, ..., w_{n-1})$,

where $w_{1..n}$ are words of a certain n-gram.

We know the Bayes' formula for computing conditional probabilities:

$P(B|A) = \frac{P(B \cap A)}{P(A)}$

We'll use an inferred equation from the Bayes' formula for our case:

$p(w_n|w_1, ..., w_{n-1}) = \frac{p(w_1,...,w_n)}{p(w_1,...,w_{n-1})}$

Translated to a human language, this asks, what is the probability of the nth word to be w_n, given the fact that the previous $n-1$ words were $w_1, ..., w_{n-1}$?
In our example, we have parsed a text from a certain training corpus, we've found that there are 100 instances of the bigram *the only*. In 75 cases out of those 100 instances this bigram was followed by the word *one*, in 10 cases by the word *reason*, in 10 cases by the word *thing* and in 5 cases by the word *known*. We can already see, that putting these values into the above formula will give us the probability estimate of what word is most likely to follow.

$p(w_3 =' one'|w_1 =' the', w_2 =' only') = \frac{p(w_1='the',w_2='only',w_3='one')}{p(w_1='the',w_2='only')} = \frac{75}{100} = 0.75$

We get the relative frequency table:

$p(w3 =' one') = 0.75$
$p(w3 =' reason') = 0.1$
$p(w3 =' thing') = 0.1$
$p(w3 =' known') = 0.05$
$p(w_3 = w_{other}) = 0.0$

We have assigned zero probability to any other word w_{other}, unseen in the training text.
Our *maximum likelihood estimate* is the probability estimate, when

using relative frequency values.

The biggest disadvantage of using MLE for probability estimates in NLP is the fact, that MLE will assign zero probabilities to any unseen data. Probabilities of longer n-grams is computed by multiplying the probabilities of shorter n-grams and so we may end up having a zero probability for an ordinary, common sentence, only because its subpart didn't occur in the training corpus. To avoid this, we can use the method called *smoothing* or *discounting* (but they mean the same thing). This method assigns very small, non-zero probabilities to data not seen in the training corpus. At the same time, the probabilities of the known data must be slightly *discounted* (decreased) so that all the probabilities add up to 100%.

2.3 Tectogrammatical Representation

In the previous chapters we were looking at statistical methods for NLP. We were able to go through some text, take the individual words, use them merely as tokens, do some calculation and at the end we obtained some probabilistic information. Lexical data - tokens - are very important for machine translation. We need to know how one word translates from one language to another, but obviously this alone is not sufficient for a good machine translation.

Team at the Charles University in Prague introduced a concept called Functional Generative Description[6]. This concept consists of three layers which capture lexical, grammatical and semantical information, relationships and dependencies between clauses. Sentences are then represented as tree structures where the words are represented by nodes with annotation labels. The layers are as follows.

m-layer : morphological layer - deals with lexical analysis, tokenization, lemmatization and assigns morphological categories to words. The output of this layer for each word is a tuple which consists of the

original form of the word, a lemma of the word and a morphological tag.

a-layer : analytical layer - presents sentences as tree structures, words are represented as nodes, captures syntactic information

t-layer : tectogrammatical layer - captures linguistic meaning of sentences. This layer also captures things which are not explicitly stated in the sentence, but are understood from the text. I.e. in the sentence *I told you yesterday...*, the t-layer would indicate that there is a topic, which was discussed yesterday. T-layer provides semantic and syntactic information, topic information as well as grammar information. Things like tenses, gender or case are covered in the t-layer.

Example of a tectogrammatical tree structure, by Cinková et al. (2009) for the sentence *A Lorillard spokewoman said: "This is an old story."* is shown in the following figure.

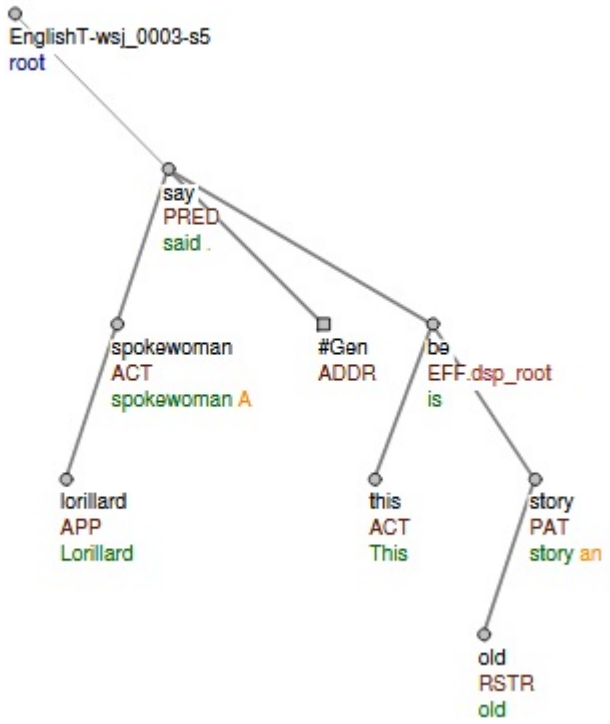


Figure 2.2: Tectogrammatical tree structure[7]

You can see several semantic labels in the Figure 2.2. They represent functions of the words in a sentence. ACT is for actor, **who** is saying something. ADDR is for addressee, actor is saying something to **someone**, here that person is unspecified - substituted by the label #gen (generalized). EFF stands for effect (**what** the actor is saying). Structuraly similar to the tectogrammatical tree structures are the UCCA trees.

2.4 Machine Translation

Previous subsections of the Chapter 2: Natural Language Processing talked about representations and models of text used for machine translation, now we will look at the Machine Translation (MT) itself. Machine translation is a process of translating a text from one human language to one or more other human languages. There are several approaches to machine translation, the basic principles of the three main ones will be discussed here.

2.4.1 Statistical MT

The most straightforward statistical MT is based on word-based and phrase-based translations. Given an input sentence, a word-based MT takes a single word and based on the conditional probability as described in the Subsection 2.2.3 (MLE) it looks for a corresponding target word. Phrase-based model on the other hand first finds a good segmentation of the source sentence into several phrases, then the translation happens for each phrase separately and at the end translated phrases are put back together with a possibility of change of their order. For training of a phrase-based MT model parallel corpora are used. Och and Ney (2000) describe alignment template approach, where alignment templates are "pairs of phrases together with an alignment between the words within the phrases". This approach, in effect, combines a word-based MT with a phrase-based

MT. To determine how good an alignment is, the Alignment Error Rate (AER) can be computed (see the subsesction 2.4.4 for the details). Current state of the art tool for phrase alignment is a tool called GIZA++[8]. Another useful tool for training translation model for a pair of any two languages (using their parallel corpus) is a "statistical machine translation system" called Moses[15]. Statistical MT covers many topics or issues which could be looked at for further investigation, such as finding the best way to split sentences into phrases, how to deal with negative verbs in different languages or looking for the best way for smoothing the model, etc.

2.4.2 Neural MT

Neural networks have become widely popular, especially since 2016 when Google started using neural machine translation in Google Translate[24]. Apart from the fact that Neural MT is structurally different to Statistical MT (building a neural network vs. creating a phrase-based model), the main difference between Neural MT and Statistical MT is that Neural MT process reads and translates a whole sentence at a time, i.e. there are no translations of individual words or phrases like in Statistical MT. It uses so-called encoder-decoder model. The principle lies in encoding the input sentence as a sequence of vectors $x = (x1, ..., x_{T_x})$ from which the context vector c is computed. The translation of the output word $\mathbf{y}$ is then predicted from the probability over the word y_T, given the context vector c and all previously predicted words for $y_T : y_1, ..., y_{t-1}$

$p(\mathbf{y}) = \prod_{t_1}^{T} p(y_t|\{y_1, ..., y_{t-1}\}, c)$ where $\mathbf{y} = (y_1, ..., y_{T_y})$

Each conditional probability then equals:

$p(y_t|\{y_1, ..., y_{t-1}\}, c) = g(y_{\{}t-1\}, s_t, c)$

where g is a nonlinear function which gives the probability of y_t and s_t is the hidden state of the recurrent neural network. The descrip-

tion of the principle of recurrent neural network MT can be found in the paper by Bahdanau D. et. al (2016).

2.4.3 Rule-based MT

This approach uses morphological and syntactic rules as well as semantic analysis of the source text and the target text. During the translation process, part-of-speech information (whether it is a verb or a noun, etc.) is collected for each word. Then each word is analyzed for obtaining syntactic information (tense, person, plural vs. singular). After that the sentence is parsed for grammatical information (subject, noun, adjective, etc.) Finally the target word is translated and its appropriate morphological form is looked up. Figure 2.3 shows the overview of the Rule-based machine translation process. Similar approach can be taken to the evaluation of the quality of MT. In fact, the core part of this thesis will be dedicated to finding relational rules which would automatically decide if the translation is good or not.

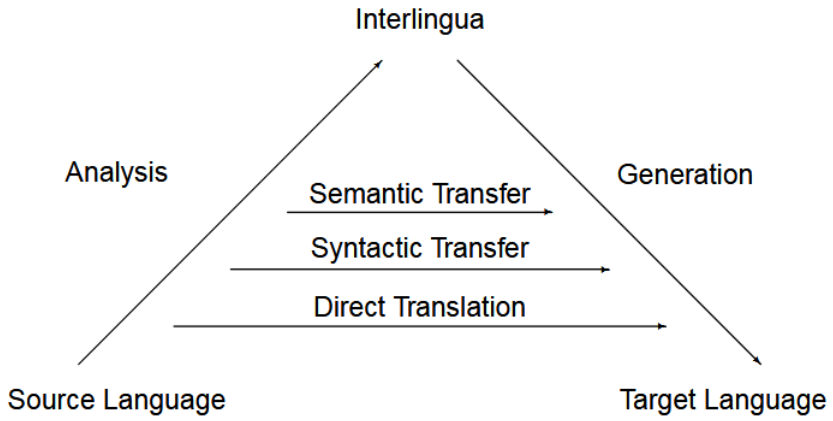


Figure 2.3: Rule-based MT scheme[5]

2.4.4 Evaluation Metrics

For our purposes we'll work mainly with the evaluation metric called HUME. There exist several other metrics for evaluating the quality of machine translation, which will be briefly described here. Some of

the metrics use the F-measure.

> F-measure - sometimes called an f-score is a statistical measure of accuracy given by the following formulas:
>
> $$F = \frac{(\beta^2 + 1) \times P \times R}{\beta^2 \times P + R}$$
>
> F ... F-measure
> P ... Precision - how many selected items are correct (relevant)
>
> $$P = \frac{\text{correct answers given}}{\text{answers given}} (\times 100\%)$$
>
> R ... Recall - how many correct (relevant) items are selected
>
> $$R = \frac{\text{correct answers given}}{\text{correct answers possible}} (\times 100\%)$$
>
> β ... a constant, using which we can change the weight of precision, so that it (precision) has higher/lower weight than recall in the given task calculation

WER - Word Error Rate
Based on the Levenshtein distance comparison, WER takes into account number of substitutions, deletions, insertions and their sum which is the same as number of words which should be correct. WER is used in machine translation as well as in speech recognition systems[14].

WMT Relative Ranking (RR) and Direct Asessment (DA)
RR and DA are metrics which were used at the Conference on Machine Translation (WMT) in Berlin in 2016, see the paper by Bojar et al.(2016). RR is a human assessment process, where the judges evaluated 5 possible machine translation outputs and ranked them best to worse. DA is also a manual evaluation, where a judge is interested in adequacy as well as fluency of the output text.

HMEANT - Human machine translation evaluation metric
HMEANT calculation, introduced by Lo C., Wu D. (2013), is based on weighted f-score. The focus is put on verbal structures, i.e. translations with omitted verbs are assigned very low score.

AER - Alignment Error Rate - Introduced by Och and Ney (2003), it is a measure used in word-based and phrase-based MT. Sometimes it is not straightforward to perform alignment between single words, for example if we have a word which can be aligned to one or more word in the target text and/or vice versa. Sometimes there might be things like idioms or function words which are not translated to the target language. So we must count with certain ambiguity there. In a manual alignment there are two kinds of alignments: S - sure alignments with no ambiguity and P - possible alignment with some ambiguity. A - then represents the quality of an alignment, computed from precision and recall, where:

$$recall = \frac{|S \cap P|}{|S|}$$

$$precision = \frac{|A \cap P|}{|A|}$$

The AER is based on F-measure calculation:

$$AER(A, P, S) = 1 - \frac{|P \cap A| + |S \cap A|}{|S| + |A|}$$

BLEU - Bilingual Evaluation Understudy
BLEU score is a value between 0 and 1. It uses modified n-gram precision to express the quality of machine translation. BLEU is a generally accepted metric. Before assigning a score to a translation, we need a reference (correct) translation. The score is computed

based on the occurence of words in a candidate translation compared to the reference translation. One disadvantage is, that shorter sentences achieve statistically better scores than longer sentences. Full description of the algorithm can be found in the paper by Papineli et al.(2002)

2.4.5 UCCA - Universal Conceptual Cognitive Annotation

Before we look at HUME evaluation metric, we need to look at UCCA, a representation used by HUME. UCCA, introduced by Abend, Rappoport (2013), is a scheme for semantic annotation. It is structured as a directed acyclic graph (DAG) where each leaf represents a word in a text. Inner nodes (non-leaves) are called units, they represent terminals or groups of words which, based on a certain semantic consideration, represent a single entity. One UCCA structure represents one interpretation of given text. Different interpretation of the same text will then result into other graph which should have the same leaves, but the structure of its internal nodes and edges will differ. This is basically one of the main purposes why UCCA has been introduced. We need the ability to express all the possible semantics of how the text may be interpreted. UCCA has multilayer structure. The lower levels in the graph focus more on the structure of the text or sentence and the relations between the arguments. Edges represent categories and show the relation between a unit and its subunit(s).

Example of UCCA:

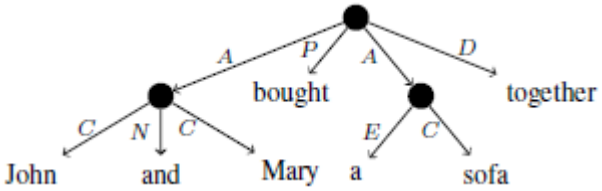


Figure 2.4: UCCA annotation graph[1]

This example was taken from the paper by Abend, Rappaport (2013) and it is shown here to give the main idea how UCCA trees

look like. More detailed specification follows.

Abend and Rappaport (2013) define a table of categories. The relation types in the table 2.1 are divided into 4 groups, based on how they relate to a "Scene". So let's take a look at what a "Scene" is. "Scene" is described as "movement or action" or a "temporally persistent state". It is meant to be used in any language and it is similar to a "clause" in language theory. A Scene typically contains one main relation, which serves as the anchor. It may contain other less important relations, which are called Adverbials. Scenes contain one or more Participants, which are either individual entities or other Scenes which play the role of a Participant. If there is a set of categories which does not qualify to be a scene, then we call those categories Centers. Non-scene set contains one or more Centers. There are several types of non-Scene units. The first one are Elaborators, which relate to a single Center. Then we have Relators, which is a semantic relation between two or more Centers. In the sentence "John and Mary bought a sofa together" two centers "John"and "Mary" are connected by a Relator "and", while the Elaborator "a" relates to a single Center "sofa". Another type are the units marked as Functions. They do not refer to an actual entity, but form merely structural pattern of part of a sentence. Example sentence: "It is OK to come late". The "It" is a part of the sentence but does not represent a specific entity, so the "It" is then marked as the Function category.

Overview of UCCA categories

Abbreviation	Category	Definition
	Scene Elements	
A	Participant	Participant in broader sense, may include location or abstract entities.
P	Process	Main relation, usually an action or movement, which evolves in time.
S	State	Main relation which does not evolve in time.
D	Adverbial	Secondary relation or temporary relation.
	Elements of Non-Scene Units	
C	Center	Categories which do not create a Scene.
N	Connector	Relation which applies to two or more Centers, and which highlights a common feature.
E	Elaborator	Relation which applies to one Center only.
R	Relator	Other type of non-Scene relation, represents either a top-down relation or a horizontal relation between two units.
	Inter-Scene Relations	
H	Parallel Scene	Links one Scene to others, if there exists semantic connection.
L	Linker	Relations between one or more Parallel Scenes
G	Ground	Relations between a speech and the Scene which has a relation to that speech.
	Other	
F	Function	A relation to an entity does not exist, but the category is required by grammatical structure.

Table 2.1: UCCA categories

4 category groups

- Scene Elements
Elements which form a Scene. They relate to an event which is specific to a certain place, time or it is a generally valid scheme (e.g. somebody likes something).

-Elements of Non-Scene Units
Elements which do have similar features, but do not form a Scene. Example of this can be enumerations, e.g.: "John, Peter and Mary" or characteristics of one entity: "top of the world".

- Inter-Scene Relations
One Scene plays the role of a Participant or Elaborator in other Scene. Example sentence: "The movie I saw yesterday was good". "The movie I saw yesterday" is a Scene which plays a role in other Scene which says that something (Participant) "was good". Other types of Inter-Scene Relations are Parallel Scenes and Linkers. These often cover temporal or conditional relations between Scenes.

- Other
Categories which do not fall into any of the groups above. At the moment there is just one category - the Function.

This categorization covers so-called Foundation Layer of UCCA structure. Abend and Rappaport go further and add additional layers to the tree structure. As an example take the sentences "He replied foolishly" and "He foolishly replied". Differences which are more subtle may be then expressed as additional layers, here to cover the adverbial scope. Adding a new layer may however change the UCCA structure of the foundation layer.

2.4.6 HUME - Human UCCA-based MT Evaluation

HUME is a semantic evaluation measure, which builds on the UCCA semantic representation scheme. Its main presumption is that the ultimate quality test of a machine translation system is going to be human judgment. Humans, however, tend to have their own views and opinions. To get a value, representing quality of a translation, which may be considered good, we'd need judgment and evaluation from more people. Also, we need to capture which parts of sentences are translated correctly and which aren't. Subsequences of sentences with regards to machine translation have so far been evaluated as either individual words or as n-grams. Now the area of our interest are semantically defined units represented by UCCA structures. HUME metric aggregates human evaluation of translation quality of

these semantic units. One of the main advantages of this approach is, that human eye is able to easily capture translations which seem to be grammatically ok, but a native speaker would never use them.

Two levels of annotation are used in HUME. Color annotation is used for atomic units, which are either individual words or phrases which in the target translation are single words/units ("Took a shower" – "dutschte").

For atomic units annotation three colors are used:

Green – Correct, the meaning of the word or the phrase has been preserved.
Orange – Partially correct, some parts of the translation are incorrect, but the overall sense has been preserved.
Red – Incorrect, the meaning of the original phrase/word has been lost.

The non-atomic units are called structural units. They consist of atomic units or/and other structural units.
For structural unit A/B annotation is used:

A – Adequate, translation is acceptable (though not necessarily perfect).
B – Bad, relations between its sub-units are incorrect/lost.

Figure 2.5 shows an example of a HUME annotation:

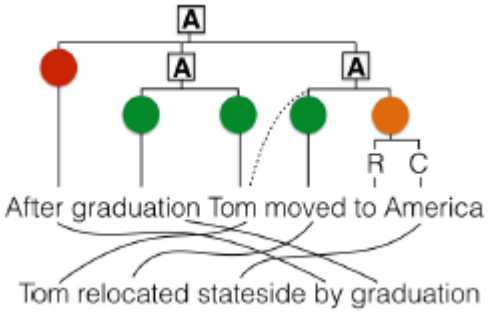


Figure 2.5: HUME annotation [3]

HUME score is then calculated as:

$$HUME(s,t) = \frac{Green(s,t) + Adequate(s,t) + 0.5 \times Orange(s,t)}{Units(s)}$$

Green(s,t), Adequate(s,t) and Orange(s,t) are the numbers of corresponding units. Units(s,t) is the number all units marked with any label. The parameters "s" and "t" refer to the source text and the target text.

3. Feature based approach

3.1 Feature vector

We need to define a feature vector, which is a set of values, which will be used to classify our data. It will be based on HUME, since it is the main metric for the evaluation of machine translation we are working with. Each sample (meaning example) will be characterised by the following attributes:

- **SentenceId** ... the sentence which the current subtree belong to

- **GreenScaled**
- **OrangeScaled**
- **RedScaled**
- **AcceptableScaled**
- **BadScaled** ... number of child nodes with Green (Orange, Red, Acceptable, Bad) HUME annotation to the total number of child nodes of a given sample

- **MissingScaled** ... number of child nodes with no annotation to the total number of child nodes. Unfortunately the input data contains some cases, where there is no annotation assigned to a child node.

- **TreeType** ... this attribute will be added as an optional attribute, it represents the tree type as defined in the subsection Basic LRNN rules.

- **numberUCCA_C**
- **numberUCCA_E**
- **numberUCCA_R**
- **numberUCCA_F**
- **numberUCCA_Other** ... number of UCCA attributes of a given type associated with child nodes.

- **class** ... finally, there is the class attribute

3.2 WEKA

WEKA - the Waikato Environment for Knowledge Analysis is a set of tools for machine learning and data mining, developed by people at the University of Waikato. It offers most of the functions for data preprocessing, classification and regression, clustering and other techniques used in machine learning.

The translation is evaluated either as A - acceptable (adequate) / B - bad or as Green / Orange / Red. So it is obvious that this will be a classification task. To demonstrate how classification in WEKA works, we'll first look at a baseline example, then we'll look for ways to make the baseline better. Final step will be to run the classification process with all our data.

3.2.1 Weka example

The input files for WEKA are in the ARFF format (Attribute-Relation File Format). It consists of the header part with the name

of the relation and the list of attributes:

```
@relation train2hume

@attribute DocumentSource string
@attribute DocumentTarget string
@attribute color green, orange, red
```

and the data part:

```
@data
"While many cases of angina can be treated with medication
, a coronary angioplasty may be required to restore the blood
supply to the heart in severe cases .", "I když většina případů
anginy pectoris může být léčena léky , pro obnovení přívodu
krve do srdce u závažných případů může být třeba koronární
angioplastika .", green
"While many cases of angina can be treated with medication
, a coronary angioplasty may be required to restore the blood
supply to the heart in severe cases .", "Mnoho případů , angina
pectoris , může být léčení s léky , koronární angioplastika
může být potřebné k obnovení přívod krve do srdce v závažných
případech .", orange
"While many cases of angina can be treated with medication
, a coronary angioplasty may be required to restore the blood
supply to the heart in severe cases .", "Mnoho obaly angina
plechovka být pečovat s lék vencový plastika může být považováno
obnovit krev přívod do srdce bouřlivý obaly.", red
```

This was part of the arff file with the training data for our baseline. The header tells us that there will be one source string, which in our case is an English sentence, and one string with the target sentence - the translation of the sentence into Czech. The final attribute is the class attribute, which represents the HUME annotation as described in the subsection dedicated to HUME and UCCA metrics.

Green is the class for the correct (reference) translation, Orange labeled text is one of the machine translations for this sentence that are available. The third sentence was created by translating each word separately, it is the worst translation and therefore belongs to the Red class. There are two other sentences in the baseline file, each with the Green, Red and Orange cases. The whole content can be found in the train2hume.arff file.

WEKA offers many classifiers, such as functions based on Bayes networks or tree algorithms (e.g. J48 - based on C4.5 decision tree, developed by Quinlan (1993)), but for our baseline we are going to use Multilayer Perceptron, this is the function which creates a neural network. Example of a neural network generated for the testcase fb-1 can be seen in the following image.

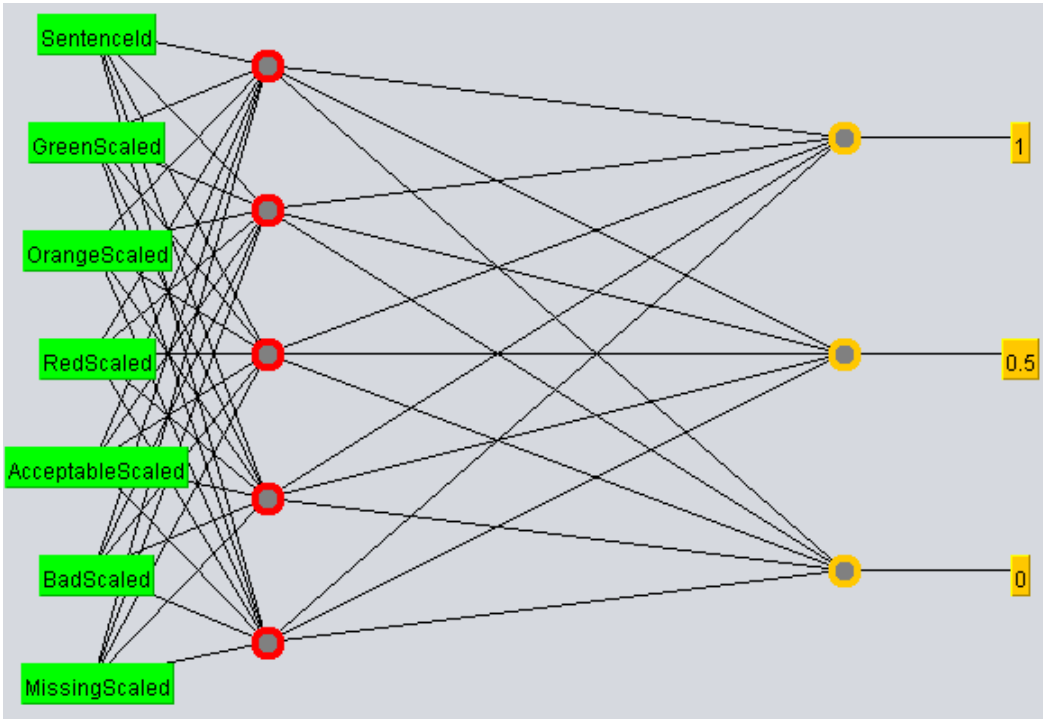


Figure 3.1: Multilayer Perceptron

4. Relational Learning and Neural Networks

4.1 LRNN - Lifted Relational Neural Networks

ANNs which are suitable for handling relational learning tasks are called Relational Neural Networks (RelNN). This is not to be confused with RNNs – recurrent neural networks, where connections between units may form cycles. RelNNs are right the opposite, they do not form cycles, their units form feedforward neural network. Uwents (2011) et.al. (page 6 – 2.2.1 Relational neural networks) states, that the output function of RelNN is implemented as a feedforward neural network.

LRNN is a concept, which has the ability to learn weights of latent relational structures. So far, none of the other existing concepts has the ability to learn weights of latent non-ground relational structures. Basically, LRNNs are templates for creating ground neural networks. Grounding means that the NN consists of only ground atoms. Ground atom contains no variables, it is basically a constant (or has only ground terms as arguments).

The word "Lifted" refers to the fact that we work with the first order logic.

Propositional logic $\rightarrow$ "LIFT" $\rightarrow$ Predicate logic (1st order logic)

Lifted models are therefore first order (predicate) logic representations from which ground models can be unfolded. With unfolding,

the number of layers in LRNN grows. LRNN doesn't define some specific ground model, but rather general set of ground models. I'll now briefly describe how LRNNs are formally defined by Šourek et al.(2015), they are based on the Herbrand model H. If N is a LRNN and $\overline{N}$ is its grounding, then:

- neuron representing a ground atom h in $\overline{N}$ is an atom neuron Ah
- neuron representing a ground fact (h,w) in $\overline{N}$ is a fact neuron $F_{(h,w)}$
- ground rules are represented by rule neurons
- neurons with activation function g_* are called aggregation neurons
- inputs of an atom neuron are aggregation neurons, atom neurons and fact neurons. Input weights are determined by the outputs of aggregation and fact neurons.
- activation functions are defined as:

$g_\wedge(b_1, ..., b_k) = sigm(\sum_{n=1}^{k} b_i - k + b_o)$

- $g_\wedge$ has high output if all of its inputs are high

$g_\vee(b_1, ..., b_k) = sigm(\sum_{i=1}^{k} b_i + b_o)$

- $g_\vee$ has high output as soon as one of its inputs is high

$g_*(b_1, ..., b_k) = \max_i b_i$

The model consists of entities, properties which an entity satisfies or not. Model also contains categories and sub categories. Each entity can be described by its soft membership to these categories.

Assumption:

- entities have properties: $HasProperty(e, p) \pm 1.0$

Ground rules:

- entities have membership in categories: $W_{ec} : IsA(e, c)$
- transitivity applies:$IsA(A, C) \leftarrow IsA(A, B), IsA(B, C)$
- entities are determined by their category:

$W'_{CeCp} : HasProperty(A, B) \leftarrow IsA(A, c_e), IsA(B, c_p),$

$HasProperty(c_e, c_p)$

The weights are learnt using SGD – Stochastic Gradient Descent. How that works is well described in the paper by Šourek et al.(2015), briefly: We have a set ε of training examples, each example is a LRNN. Our aim is to find weights within each of these LRNNs so that the cost of training individual atoms is minimized. Algorithm iterates in random order over all LRNNS and updates weights by gradient descent steps given some learning rate.

4.2 LRNN rules introduction

We are going to define rules, based on which the LRNN network will classify the input data. Consider the most basic example: we define rules, which say, that if there is a node Y labeled as green or red, and a node X which is a parent node of Y, then the root node X will be classified either as "acceptable" or as "bad":

a1(a) :- green(Y), parent(X,Y).
a2(a) :- red(Y), parent(X,Y).
0.0 finalKappa(a) :- a1(a).
0.0 finalKappa(a) :- a2(a).

The value finalKappa(a) is the value of the output neuron. The settings "0.0 finalKappa(a) :- ..." means that the LRNN framework will set the initial value of finalKappa as a random value (0.0). Then follows the process of training of the network. Once the network is fully trained we will be able to classify each tree either as 1.0 - "acceptable" or as 0.0 - "bad". The input data have the following structure:

Rules and results:
0.0 red(n16), parent(n15,n16).
1.0 green(n17), parent(n15,n17).

```
Example #0;
expected:  0.0; actual:  0.0
Example #1;
expected:  1.0; actual:  1.0
```

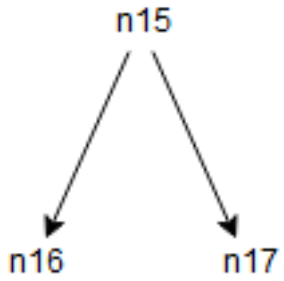


Figure 4.1: tree 1

As expected, each of these two examples has been classified to be in a different class.

Other, more complex, tree:

0.0 red(n16), parent(n15,n16).
1.0 green(n17), parent(n15,n17).
1.0 green(n18), parent(n15,n18).
1.0 green(n19), parent(n15,n19).

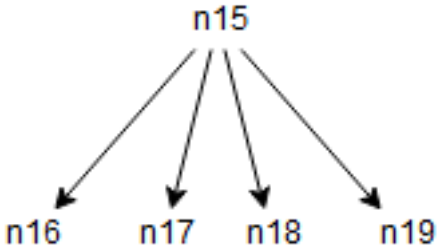


Figure 4.2: tree 2

For better clarity, the results are put into the table 4.1 and sorted top-down by the example #. (Normally, the examples on the output are in random order, since the input data are, by default, shuffled.) The first column is the example number. The second column, Expected, is the expected value, which is the leftmost value of the input data (i.e. for the row with red(n16) this equals 0.0, for the rest it is 1.0). The third column, Classified, is the class. Actual, is the computed value of the output neuron, which is the fourth column. The last column, Threshold, is the computed threshold based on which the data is classified to be either 0.0 or 1.0.

Results:

Example #	Expected	Classified	Actual	Threshold
0	0.0	0.0	1.387778E-17	0.5
1	1.0	1.0	1.0	0.5
2	1.0	1.0	1.0	0.5
3	1.0	1.0	1.0	0.5

Table 4.1: tree 2

Again, the actual classification meets the expected classification. What happens if we change one of the examples with a green node from 1.0 to 0.0 (in the last row):

0.0 red(n16), parent(n15,n16).
1.0 green(n17), parent(n15,n17).
1.0 green(n18), parent(n15,n18).
0.0 green(n19), parent(n15,n19).

Results:

Example #	Expected	Classified	Actual	Threshold
0	0.0	0.0	0.045264	0.36516
1	1.0	1.0	0.685055	0.36516
2	1.0	1.0	0.685055	0.36516
3	0.0	1.0	0.685055	0.36516

Table 4.2: tree 2

LRNN has enough correctly classified examples for training, so the Example #3 in the table 4.2 has been classified to be 1.0, no matter the change. Let's revert the change and instead let's try to change the expected value of the example with the red node.

1.0 red(n16), parent(n15,n16).
1.0 green(n17), parent(n15,n17).
1.0 green(n18), parent(n15,n18).
1.0 green(n19), parent(n15,n19).

Example #	Expected	Classified	Actual	Threshold
0	1.0	1.0	1.0	1.0
1	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0

Table 4.3: tree 2

Table 4.3 shows, that even the example with the red node has been classified to be 1.0. There are not enough data with the red nodes, in fact none, for training of our network, which is why LRRN will not know that they should be classified as 0.0. So we cannot say that LRNN has classified Example #0 incorrectly.

Next example will be a little bit more complex. In Figure 4.3 you can see a tree with leave nodes n12, n14, n15, n17, n18. Each of these leave nodes will have assigned a HUME color.

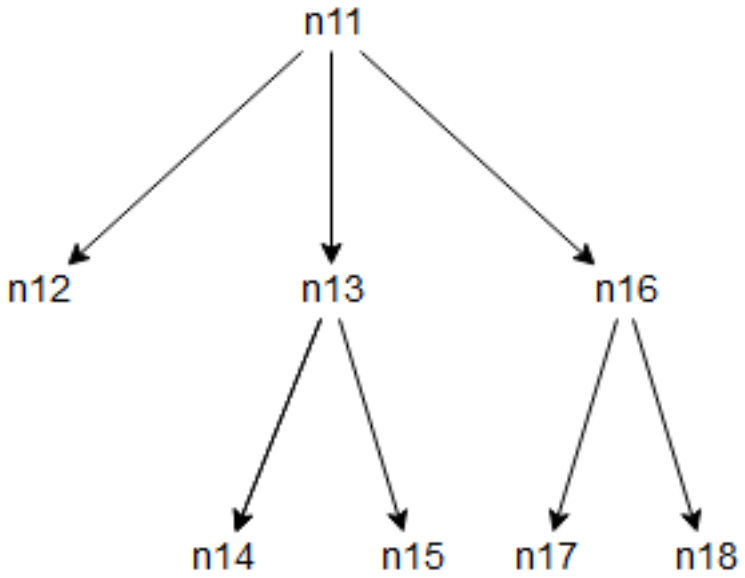


Figure 4.3: tree 3

Input data, which can be found in the file examples3b.txt, correspond to the table 4.4. For the sake of simplicity there are only red

leaves (R) and green leaves (G).

		leave nodes				
Ex.#	class	n12	n14	n15	n17	n18
0	1.0	G	G	G	G	G
1	0.0	R	R	R	R	R
2	1.0	G	G	R	G	R
3	0.0	G	G	G	R	R
4	1.0	R	G	R	G	G
5	0.0	G	R	R	G	R
6	1.0	R	G	R	G	R
7	1.0	R	R	G	R	G

Table 4.4: tree 3 - leave nodes

Rules (rules3.txt):

```
a1(a) :- green(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
acc(X) :- green(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
a2(a) :- red(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
acc(X) :- red(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
a3(a) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
a4(a) :- red(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
0.0 finalKappa(a) :- a1(a).
0.0 finalKappa(a) :- a2(a).
0.0 finalKappa(a) :- a3(a).
0.0 finalKappa(a) :- a4(a).
```

Results:

Example #	Expected	Classified	Actual	Threshold
0	1.0	1.0	1.108886	0.351598
1	0.0	0.0	0.020673	0.351598
2	1.0	1.0	0.682522	0.351598
3	0.0	1.0	0.682522	0.351598
4	1.0	1.0	0.682522	0.351598
5	0.0	1.0	0.682522	0.351598
6	1.0	1.0	0.682522	0.351598
7	1.0	1.0	0.682522	0.351598

Table 4.5: tree 3

Class 1.0 is an acceptable translation - HUME label A, class 0.0 is a bad translation - HUME label B.
2 out of the 8 examples have been classified incorrectly. #5 and #3 belong to the class 0.0, but LRNN has classified them as 1.0. These two misclassified examples are very similar. They both have the green node n12 and one of their subtrees has both children red. The difference is in the other subtree where #3 has two green nodes and #5 has one green node and one red node. If we look at the rules again, you will find that there is no rule which would describe a situation where the children of a subtree would have different color. So we need to add a new rule (rules3b1.txt):

a5(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2).

Results of #3 and #5 are now:

Example #	Expected	Classified	Actual	Threshold
3	0.0	0.0	0.011725	0.412676
5	0.0	1.0	0.813626	0.412676

Table 4.6: tree 3

Example #3 is now correctly classified as 0.0. Example #5 is still classified incorrectly. We would need to increase the number rules for #5 to be classified correctly.

4.3 LRNN for machine translation evaluation

Main task of this thesis is to create a system which would be able to automatically evalute machine translation on a sentence level. In other words, we are interested in the value (A or B) of the root node of a give sentence.
In this section we'll look at the situations which may occur within the subtrees of our input data. Root node is represented here by the variable X, child nodes by the variables Y1,...,Yn. The cases are named based on the child node HUME labels. What needs to be emphasized is the fact, that to have a good classifier, rules do not need to cover every possible situation in the subtrees of our input data. Weight learning of a good and to some extent generic neural network is also important for good classification results.

4.3.1 Basic LRNN rules

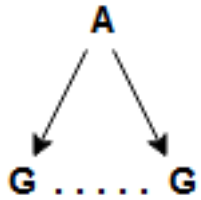
Diagram and LRNN rules	**Description**
case G..G:	
	Subtree with two or more child nodes labeled as green. Logically the root of the subtree should be A.

Figure 4.4: case G..G

```
a1(a) :- green(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
acc(X) :- green(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
0.0 finalKappa(a) :- a1(a).
```

The reason why there is the same rule twice, each time with a different left side, is that the root node may serve either as a root of

the whole sentence, which is the first case `aGG(a)` or it may be an internal node and serve as a child node of some tree hierarchically higher, which is the second case `acc(X)`.

case G:

Single child node labeled as green is considered A.

Figure 4.5: case G

```
a1(a) :- green(Y1), parent(X,Y1).
acc(X) :- green(Y1), parent(X,Y1).
0.0 finalKappa(a) :- a1(a).
```

case A G:

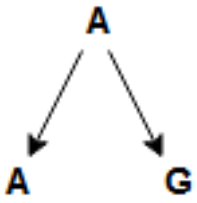


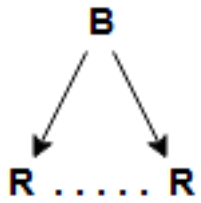
Figure 4.6: case A G

Subtree with two children, where one of them is A, is considered to be A. Even if the other node wasn't green, the subtree would still be considered as acceptable. The reason being, leaf labeled as A represents correct translation of part of a sentence. Such correctly translated sequence of words, concatenated with a single, possibly incorrectly translated word, would still be considered as acceptable. Two or more incorrectly translated words might, however, not be considered to be A.

```
a1(a) :- acc(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
accx1(X) :- acc(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
...
...
accxn(X) :- [some rules]
acc(X) :- accx1(X).
...
...
acc(X) :- accxn(X).
0.0 finalKappa(a) :- a1(a).
```

Here is the situation, which we talked about previously with the `case G..G`. The left child node is labeled as A, but we do not know how the underlying subtree look like, so the rules must be amended to be able to cope with different scenarios.

case R..R:



Two or more child nodes labeled as red are considered to be B, bad translation.

Figure 4.7: case R..R

```
a1(a) :- red(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
acc(X) :- red(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
0.0 finalKappa(a) :- a1(a).
```

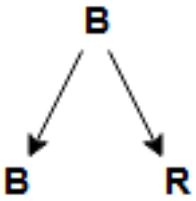
case R:

Single child node labeled as red is considered B.

Figure 4.8: case R

```
a1(a) :- red(Y1), parent(X,Y1).
acc(X) :- red(Y1), parent(X,Y1).
0.0 finalKappa(a) :- a1(a).
```

case B R:



Subtree with two children, one is B, the other is R. The root node should be classified then as B, bad translation. The rationale behind this is similar to the `case A G`.

Figure 4.9: case B R

```
a1(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accx1(X) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
..  see Appendix A
```

`Note:` Since in many cases the rules are quite similar, there is no need to copy and paste many lines of text, where the difference will

be only in one or two lines. In such situations, there will be a note ".. see Appendix A". You will find only the most important bits of the rules stated here, whereas in Appendix A you will find the complete list of all the rules for all the cases.

case G O:

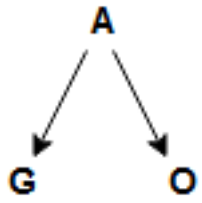


Figure 4.10: case G O

The orange child node represents either a single orange leaf or a subtree with two or more orange leaves. Such a situation should be classified as A. If there exists a green leaf and there is no red leaf, then the subtree is acceptable. Internal orange nodes within a properly trained LRNN network should ideally have intrinsic value of 0.5.

```
a1(a) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accx1(X) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
..  see Appendix A
```

4.3.2 Extended LRNN rules with UCCA annotations

So far the rules were based on the parent-child relations between a root node and a child node, labeled by one of the three colors, G - green, O - orange, R - red or by the labels A - acceptable or B - bad. We have enough rules for creating a good LRNN network for evaluation of machine translation. There might be cases however, which are not so common. To be able to deal with such less frequent situations, we will need to define more specific rules. Obviously the aim is not to define rule for everything, that would be impossible. Our scheme will be extended by UCCA annotation of the red node. This will be introduced as an option in certain situations, so there

will be a choice between standard rules (based on HUME labels only) and extended rules with UCCA annotations, as described by Abend O. and Rappoport A. (2013).
UCCA annotations which may have a negative impact on the translation are mainly:
C (Center) - as described in the subsection 2.4.5, Centers are those units which do not create a scene, therefore if there are two children connected via a root node where one child is labeled as C, then, even though these two child nodes are not completely unrelated, their linguistic and semantic bond might be somewhat lower then if they were part of the same scene.
E (Elaborator) - also applies to non-scene units, more specifically to a single Center. So E annotation may have similar impact as C annotation.
R (Relator) - links two non-scenes units either on the same hierarchical level or top-down level, Similar impact as C annotation.
F (Function) - this annotation is used in cases where certain word in a sentence is present only because the grammatical requires it to be there. So if the translated word is there, but labeled as F, then it might indicate that something is not right.

case G R:

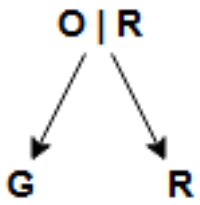


Figure 4.11: case G R

If one child node is green and the other one is red, then it seems logical, that the root node will be classified as orange. However the presence of certain UCCA labels might be the factor why this would be classified rather as red. (We are talking here about only two leaf nodes, which is why R - red classification is more suitable then B - bad.)

```
a1(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accx(X) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
0.0 finalKappa(a) :- a1(a).

extended UCCA rules:

a1(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccac(Y2).
a2(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccae(Y2).
a3(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccar(Y2).
a4(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccaf(Y2).
accx1(X) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
uccac(Y2).
accx2(X) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
uccae(Y2).
accx3(X) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
uccar(Y2).
accx4(X) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
```

```
uccaf(Y2).
acc(X) :- accx1(X).
acc(X) :- accx2(X).
acc(X) :- accx3(X).
acc(X) :- accx4(X).
0.0 finalKappa(a) :- a1(a).
0.0 finalKappa(a) :- a2(a).
0.0 finalKappa(a) :- a3(a).
0.0 finalKappa(a) :- a4(a).
```

case O R:

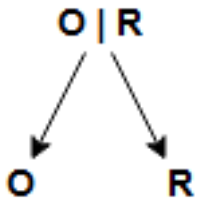


Figure 4.12: case O R

Since O node may represent more than one orange node, whereas the R node represents only one red node (it would have been labeled as B if it covered more than one red node), then we can hesitantly conclude, that the outcome is going to be O. As in the previous case G R, presence of certain UCCA annotation for the red node could be the reason why the outcome will be classified as R. Again, both version, with and without UCCA rules, are stated here.

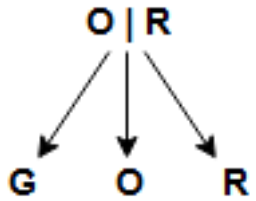
```
a1(a) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accx(X) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
0.0 finalKappa(a) :- a1(a).

extended UCCA rules:

a1(a) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
uccac(Y2).
a2(a) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
```

```
uccae(Y2).
...
accx1(X) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
uccac(Y2).
accx2(X) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
uccae(Y2).
..  see Appendix A
```

case G O R:



One green node, one red node and one or more orange nodes, this should be classified as orange. Presence of C, E, R or F UCCA annotations for the red node may be the reason why this will be classified as R.

Figure 4.13: case G O R

```
a1(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
red(Y3), parent(X,Y3).
accx(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
red(Y3), parent(X,Y3).
0.0 finalKappa(a) :- a1(a).

extended UCCA rules:

a1(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
red(Y3), parent(X,Y3), uccac(Y3).
a2(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
red(Y3), parent(X,Y3), uccae(Y3).
...
accx1(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
red(Y3), parent(X,Y3), uccac(Y3).
accx2(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
```

```
red(Y3), parent(X,Y3), uccae(Y3).
..  see Appendix A
```

case A O R:

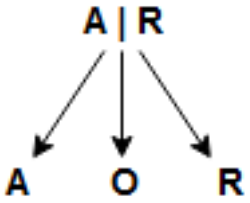


Figure 4.14: case A O R

This subtree consists of two or more green nodes (child A), one or more orange nodes and one red node. If it wasn't for the red node, this subtree would be classified as A. With the single red node, it may still be classified as A, for example if A represents a subtree with let's say 5 words, where all of the are G, in such a situation one incorrectly classified word should still be OK for acceptable translation of the whole sentence. So we can't really know. The idea here is to distinguish between an ordinary red node and a red node, which is really a bad red node. If the red node has a UCCA annotation C or E then this might be the case. As opposed to previous cases (such as case G O R), UCCA annotations R and F are not used here, since we only look for extreme situations where a single red node would indicate bad translation of otherwise acceptable subtree.

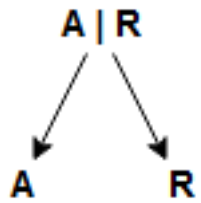
```
a1(a) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
red(Y3), parent(X,Y3).
accx1(X) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
```

```
red(Y3),
parent(X,Y3).
..  see Appendix A

extended UCCA rules:

a1(a) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
red(Y3), parent(X,Y3), uccac(Y3).
a2(a) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
red(Y3), parent(X,Y3), uccae(Y3).
accx1(X) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
red(Y3), parent(X,Y3), uccac(Y3).
accx2(X) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
red(Y3), parent(X,Y3), uccae(Y3).
..  see Appendix A
```

case A R:



Similar case to the previous one, should be classified as A unless the red node has UCCA annotation C or E.

Figure 4.15: case A R

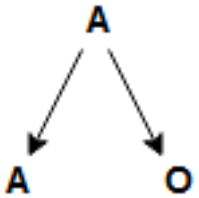
```
a1(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accx1(X) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
..  see Appendix A

extended UCCA rules

a1(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccac(Y2).
a2(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccae(Y2).
accx1(X) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
```

```
uccac(Y2).
accx2(X) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
uccae(Y2).
..  see Appendix A
```

case A O:

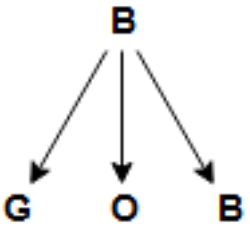


One A node (i.e. two or more green nodes), at least one orange node, this case is quite clear, will be classified as A.

Figure 4.16: case A O

```
a1(a) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
accx1(X) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
..  see Appendix A
```

case G O B:



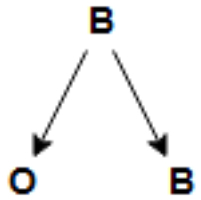
If part of a sentence is classified as bad translation, then the whole sentence can't be classified as acceptable. Therefore this subtree should be classified as B.

Figure 4.17: case G O B

```
a1(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
acc(Y3), parent(X,Y3).
accx1(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
```

```
acc(Y3), parent(X,Y3).
..  see Appendix A
```

case O B:

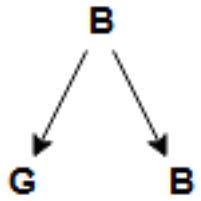


Similar situation as `case G O B`, subtree should be classified as B.

Figure 4.18: case O B

```
a1(a) :- orange(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accx1(X) :- orange(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
..  see Appendix A
```

case G B:

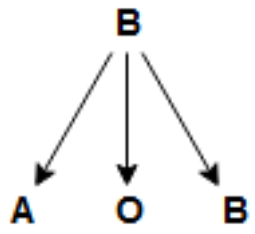


Similar situation as the previous case, subtree should be classified as B.

Figure 4.19: case G B

```
a1(a) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accx1(a) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
..  see Appendix A
```

case A O B:



Similar situation as the `case G O B`, if part of a sentence is classified as B then the whole sentence can't A anymore, therefore B.

Figure 4.20: case A O B

```
a1(a) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
acc(Y3),
parent(X,Y3).
accx1(a) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),
acc(Y3), parent(X,Y3).
..  see Appendix A
```

5. Experiments

5.1 Input data description

The input data, which we have available were provided by the scientists at the Charles University in Prague. We have machine translation from English to four languages - Czech, German, Polish and Romanian. For each language we have more than 500 sentences represented as a tree structure with HUME-annotated leaves. Some of the sentences are present twice, but in those cases, the HUME and UCCA annotations were done by two different annotators. In total, we have more than 2200 sentences. Each sentence has around 15 - 20 subtrees, that makes up to 10 000 examples for each language. For each sentence we have the source language (English), the actual machine translation and a reference (correct) translation in the target language. The data were provided in two Excel spreadsheets, they can be found on the CD attached to this theses.

sentencesUTF8.xlsx - contains all the sentences
nodes2UTF8.xlsx - contains all the nodes with references to parent and child nodes (tree structure), UCCA annotations and HUME annotations.

Before running any experiments, the data had to be parsed and converted into the correct format for the input for LRNN framework. The code of the parser can be found on the CD, should there be the need to reuse it for future work.

5.1.1 Example of the input data

The first sentence of our input data is:
"While many cases of angina can be treated with medication , a coronary angioplasty may be required to restore the blood supply to the heart in severe cases ." This data has already be preprocessed, the comma after the word "medication" and the period after the word "cases" have been separated by a space.
Following picture shows a HUME annotated tree of part of the first sentence of the input data.

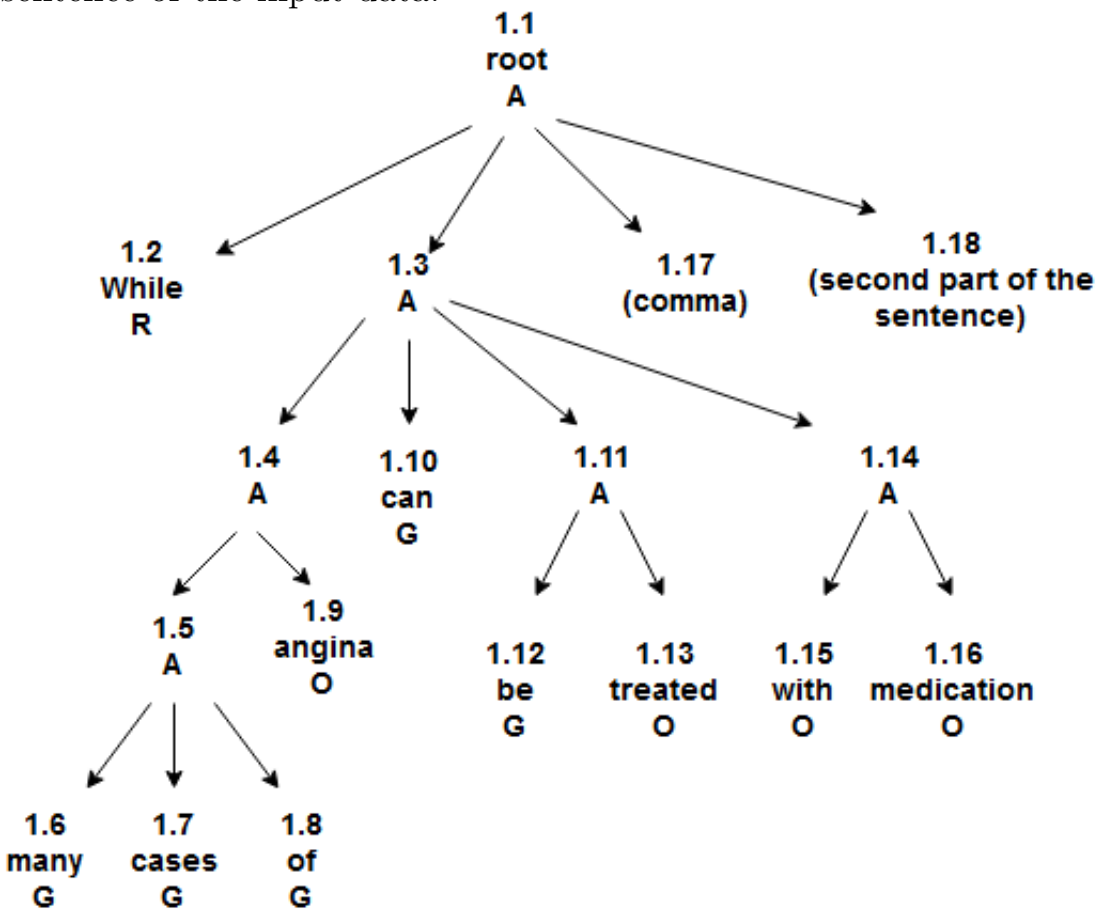


Figure 5.1: HUME annotated tree

Subset of training data corresponding to the tree in Figure 5.1:

- Example of the format of the input data for the feature-based approach:

@attribute SentenceId numeric
@attribute GreenScaled numeric
@attribute OrangeScaled numeric
@attribute RedScaled numeric
@attribute AcceptableScaled numeric
@attribute BadScaled numeric
@attribute MissingScaled numeric
@attribute TreeType numeric
@attribute class 1, 0.5, 0

@data
505, 0.0, 0.0, 0.2, 0.2, 0.2, 0.0, 16, 0
505, 0.5, 0.5, 0.0, 0.0, 0.0, 0.0, 6, 1
505, 0.5, 0.0, 0.5, 0.0, 0.0, 0.0, 7, 1
505, 0.2, 0.0, 0.0, 0.8, 0.0, 0.0, 3, 0
505, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1, 1
...
etc.
The meaning of the individual attributes was fully described in the section 3.1 Feature vector.

- Example for relational-based approach. (E.g. for red(n12_505) the n12 is the node number, 505 refers to the sentence Id.)

1.0 red(n12_505), parent(n11_505,n12_505), acc(n13_505),
parent(n11_505,n13_505), acc(n118_505), parent(n11_505,n118_505).
1.0 green(n112_505), parent(n111_505,n112_505),
orange(n113_505), parent(n111_505,n113_505).
1.0 orange(n115_505), parent(n114_505,n115_505),
orange(n116_505), parent(n114_505,n116_505).
1.0 acc(n14_505), parent(n13_505,n14_505),
green(n110_505), parent(n13_505,n110_505),
acc(n111_505), parent(n13_505,n111_505),
acc(n114_505), parent(n13_505,n114_505).
1.0 acc(n15_505), parent(n14_505,n15_505),

orange(n19_505), parent(n14_505,n19_505).
1.0 green(n16_505), parent(n15_505,n16_505),
green(n17_505), parent(n15_505,n17_505),
green(n18_505), parent(n15_505,n18_505).
...
etc.

5.1.2 Observed values

Values to be observed are:

1. train error
Train error is the ratio of correctly classified examples to all examples on the training data set

2. test error
Test error is the ratio of correctly classified examples to all examples on the testing data set, in our case the test folds.

3. test MSE Mean Square Error is defined as:

$MSE = \frac{1}{n}\sum_{i=1}^{n}(e_{1i} - e_{2i})^2$,

where e_{1i} is the expected value of an example i and e_{2i} is the label of an example i.

4. test AUC pr
In the subsection 2.4.4 Evaluation Metrics, precision and recall were explained. Basically, precision tells us how accurate the model is and recall represents the completeness of returned data. Usually, more precision means lower recall and vice versa. Based on the practical application for what the classifier is used, we may want either high precision and then recall would be less important, or we tolerate some inaccuracy to get more data to be in our class. Obviously the best case is if we have a model with high precision and high recall. So that's the information which we can visually get

from the precision-recall (pr) curve. The closer the curve is to the top-right corner the better. In figure 5.2, the AUC is the grey area under the red pr curve.

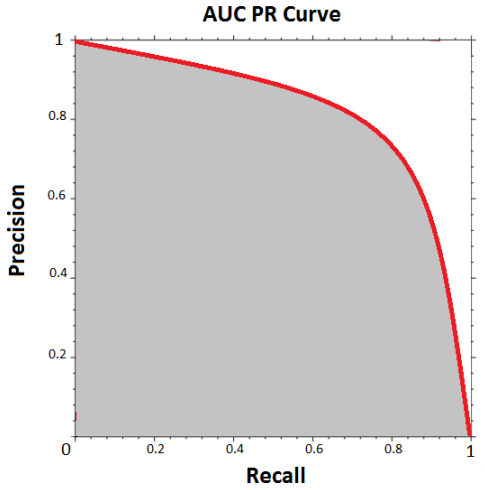


Figure 5.2: AUC PR Curve

5. test AUC roc

Area Under Curve ROC evaluates a classifier how well it is able to separate the classes. ROC (Receiver Operating Characteristic) curve is a plot of all classification thresholds (between 0 and 1) of a binary classifier. True Positive Rate (Y axis) is the ratio of correctly classified positive examples to the total number of positive examples. False Positive Rate (X axis) is the ratio of incorrectly classified negative examples (i.e. classified as positive when they should be classified as negative) to the total number of negative examples. A good classifier has a ROC curve close to the top left corner and AUC close to 1, whereas a bad classifier has a ROC curve close to the diagonal and AUC close to 0.5. The diagonal itself represents a classifier which has no better results than random guessing. See the figure 5.3, the ROC curve is in red, the AUC is the area under the ROC curve.

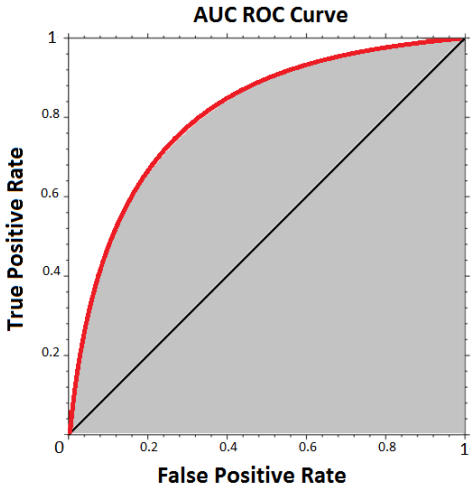


Figure 5.3: AUC ROC Curve

5.2 Testcases and results - feature-based

Feature-based approach will serve as a baseline for future comparison of the relational-based approach. For classification we will use the function of the WEKA framework called `MultilayerPerceptron`, which is how a neural network is called in WEKA. For the test values, cross-validation will be used using 10 folds.

5.2.1 Testcase fb-1

We will start with the most basic testcase fb-1 (feature-based - 1). The five observed values are defined in the subsection 5.1.2 - Observed values. The attributes for the classifier are described in the section 3.1 - Feature vector. We are not going to use the TreeType attribute in this testcase. The input data can be found in the files:

Data files: examplesWeka{Cs,De,Pl,Ro}_All.arff

Language	train error	test error	test MSE	test AUC pr	test AUC roc
Czech	0.174766	0.187633	0.286	0.870	0.855
German	0.212191	0.213174	0.3122	0.829	0.840
Polish	0.278534	0.285377	0.354	0.764	0.821
Romanian	0.137366	0.138276	0.2692	0.884	0.908

Table 5.1: Testcase fb-1

5.2.2 Testcase fb-2

In this testcase the input data will be enriched by the attribute TreeType. The value represents all the different tree types as described in the section 4.3 - LRNN for machine translation evaluation.

Data files: examplesWeka{Cs,De,Pl,Ro}_TreeType_All.arff

Language	train error	test error	test MSE	test AUC pr	test AUC roc
Czech	0.180887	0.188007	0.2864	0.871	0.853
German	0.213502	0.213502	0.3125	0.829	0.841
Polish	0.270143	0.287332	0.3532	0.767	0.822
Romanian	0.137253	0.138276	0.2688	0.883	0.910

Table 5.2: Testcase fb-2

From both testcases we can see that the results are quite similar. Romanian language was the one with the most correctly classified instances. The values of AUC roc curve are all above 80%, which can be considered to be an average, fairly good result.

5.2.3 Testcase fb-3

For the testcase fb-3 the input data from the previous testcase will be enriched with 5 more attributes, which say how many UCCA annotation of the type C, E, R, F, or other are associated with child nodes of the current sample (subtree). Since the UCCA annotation capture additional semantic information, we should be getting better results.

Data files: examplesWeka{Cs,De,Pl,Ro}_TreeType_UCCA_All.arff

Language	train error	test error	test MSE	test AUC pr	test AUC roc
Czech	0.178264	0.183885	0.2865	0.875	0.857
German	0.178928	0.196461	0.3052	0.855	0.861
Polish	0.265336	0.278289	0.3502	0.779	0.830
Romanian	0.135092	0.136002	0.2668	0.886	0.912

Table 5.3: Testcase fb-3

From the results we can see, that in all cases the train and test error values are better. For Czech and Romanian the improvements are

not significant, for Polish the improvement is between 0.5% - 1%., but for German, there is an improvement of more than 1.5% for the test error, which is nonnegligible (not talking about the improvement of the train error of 3.5%). The AUC pr/roc values are all mostly the same, slightly better.

5.3 Testcases and results - relational-based

We will conduct several testcases with different rules sets and different input data. We will then look into the values obtained and try to determine which set of rules are the most suitable for the evaluation of machine translation and why. For our experiments three different sets of LRNN rules will be used. The first set of rules will be the set as described in the subsection 4.3.1, plus the cases G R and O R without UCCA annotations. The second set will cover all cases (i.e. subsection 4.3.1 and 4.3.2 combined) without UCCA annotations. The third set will cover all cases plus UCCA annotations where applicable.

a) Set of Basic LRNN rules This set consists of cases: G G, G, A G, R R, B R, G O, G R, O R.

b) Set of Extended LRNN rules This set consists of all the cases from set a) plus cases: G O R, A O R, A R, A O, G O B, O B, G B, A O B.

c) Set of Extended LRNN rules with UCCA annotations This set consists of all the cases from set b) extended with UCCA annotations. For the complete list of all three sets see Appendix B - Sets of LRNN rules for experiments.

Experiments are run with the settings:
- number of folds: 10
- number of learning steps: 1000

5.3.1 Testcase 1

The first testcase 1 (relational-based - 1), will be based on generic rules with no rules based on HUME or UCCA annotations. The complete set of generic rules can be found in Appendix B or on the CD:

Data files: examples{Cs,De,Pl,Ro}_All.txt
LRNN rules file: rulesGeneric.txt

Language	train error	test error	test MSE	test AUC pr	test AUC roc
Czech	0.202347	0.20825	0.162679	0.807008	0.638639
German	0.28255	0.284754	0.218365	0.68547	0.58586
Polish	0.333804	0.405052	0.235353	0.583616	0.575971
Romanian	0.237466	0.261774	0.212323	0.688104	0.642454

Table 5.4: Testcase 1

We have obtained some results, in the next testcases we will see if we are able to obtain better results if the LRNN rules are more specific.

5.3.2 Testcase 2

For this testcase, we will use the Basic rules set. First we'll examine how the results change with increasing number of input data using 5 batches of English-to-Czech translations. We'll start with **100** sentences for the first batch and then we'll add 100 sentences more with each batch, until the fifth batch, which will have **all** the sentences available, which is over 500 (for English-to-Czech it is 529).

Data files: examplesCs_**100**.txt (100 sentences),....,
examplesCs_**All**.txt (529 sentences)
LRNN rules file: rulesBasic.txt

Sentences	train error	test error	test MSE	test AUC pr	test AUC roc
100	0.190704	0.196732	0.155352	0.824688	0.689880
200	0.187123	0.194081	0.145393	0.841873	0.705534
300	0.200836	0.205088	0.154387	0.826062	0.687091
400	0.202979	0.208824	0.156933	0.823731	0.694995
529	0.19825	0.206499	0.155815	0.829746	0.691603

Table 5.5: Testcase 2a

We can see that with increasing number of examples, the train error, test error and test MSE all follow the same characteristic. For 300 and 400 sentences we got worse values than for 100 and 200 sentences, but when all the sentences (529) were used, the results tend to get better again. It seems that after certain threshold, the errors tend to go down and the AUC to go up again, unfortunately, we are limited on the data set, to make this conclusion. In the second part of testcase 2 the results for all four target languages will be compared. Following table shows the results for all the input data (i.e. over 500 sentences) for a given language.

Data files: examples{Cs,De,Pl,Ro}_All.txt
LRNN rules file: rulesBasic.txt

Language	train error	test error	test MSE	test AUC pr	test AUC roc
Czech	0.19825	0.206499	0.155815	0.829746	0.691603
German	0.276666	0.284754	0.222829	0.685741	0.584489
Polish	0.32751	0.395762	0.255499	0.587642	0.580916
Romanian	0.215585	0.243799	0.192296	0.692827	0.655691

Table 5.6: Testcase 2b

We can see that the results for all four language show slightly better results, than the all-generic rules from testcase 1. The biggest improvements are shown for Czech language, the train error got below 20%, the AUC pr increased by 3% and the AUC roc is up by 6%, which is considered significant improvement.

5.3.3 Testcase 3

For this testcase the network will be learned using the set of extended rules (without UCCA annotations, can be found in Appendix B). The input data are going to be the same, so we'll be able to compare the results with those of the testcase 2b.

Data files: examples{Cs,De,Pl,Ro}_All.txt
LRNN rules file: rulesExtended.txt

Language	train error	test error	test MSE	test AUC pr	test AUC roc
Czech	0.198249	0.205625	0.151446	0.842747	0.703289
German	0.276958	0.281311	0.226183	0.683834	0.580093
Polish	0.331721	0.393643	0.248456	0.590491	0.582195
Romanian	0.215585	0.243799	0.193729	0.694989	0.659354

Table 5.7: Testcase 3

From the comparison with the testcase 2, it is clear that the results are not much different, but we can see small improvements, such as the AUC roc is slightly better for 3 out of 4 languages by around 0.5 to 1%.

5.3.4 Testcase 4

For testcase 4 the set of extended LRNN rules with UCCA annotations (see Appendix B) will be used. The goal of this testcase is to determine the impact of extending the rules set from the testcase 3 with additional relations as described in the subsection 4.3.2.

Data files: examples{Cs,De,Pl,Ro}_Ucca_All.txt
LRNN rules file: rulesExtendedUcca.txt

Language	train error	test error	test MSE	test AUC pr	test AUC roc
Czech	0.194138	0.203125	0.174442	0.823884	0.683836
German	0.270291	0.277377	0.24451	0.679292	0.56758
Polish	0.327338	0.399103	0.301993	0.584006	0.563956
Romanian	0.217393	0.246643	0.21467	0.686908	0.647577

Table 5.8: Testcase 4

As we can see, the train error is, by another 0.5 % better than the previous testcase, the test error, however, hasn't improved. The MSE, AUC pr and AUC roc got somewhat worse, therefore we need to look for other ways how to obtain better model.

5.3.5 Testcase 5

In this testcase we'll look at whether we'll be able to obtain better model using input data for combination of two or more languages.

Data files: examplesCs{De,Pl,Ro}_All.txt
LRNN rules file: rulesBasic.txt

Language	train error	test error	test MSE	test AUC pr	test AUC roc
Czech & German	0.237486	0.246453	0.18334	0.769111	0.648975
Czech & Polish	0.278703	0.320167	0.207284	0.687497	0.631804
Czech & Romanian	0.215366	0.232162	0.179079	0.76211	0.679215

Table 5.9: Testcase 5a

Comparing the values from Testcase 5a with the values from Testcase 2b, we can see, that if we combine input data of two languages, the results correlate to the arithmetic mean of the individual results of these two languages. Combining the input data for all four languages will give following results. Neither the combination of all four languages gives better results than the individual language input data.

Data files: examplesCsDePlRo_All.txt
LRNN rules file: rulesBasic.txt

Language	train error	test error	test MSE	test AUC pr	test AUC roc
Cze, Ger, Pol, Rom	0.267131	0.296986	0.215714	0.686546	0.623902

Table 5.10: Testcase 5b

Data files: examplesCsDePlRo_All.txt
LRNN rules file: rulesExtended.txt

Language	train error	test error	test MSE	test AUC pr	test AUC roc
Cze, Ger, Pol, Rom	0.267358	0.296986	0.213771	0.688935	0.623135

Table 5.11: Testcase 5c

Data files: examplesCsDePlRo_Ucca_All.txt
LRNN rules file: rulesExtendedUcca.txt

Language	train error	test error	test MSE	test AUC pr	test AUC roc
Cze, Ger, Pol, Rom	0.268789	0.301052	0.241068	0.68724	0.612666

Table 5.12: Testcase 5d

5.3.6 Testcase 6

So far the best results were for the setting in testcase 3 - extended rules without UCCA annotations. Second best in terms of error values was testcase 4 - extended rules with UCCA annotations and in terms of AUC values testcase 2 - basic rules. So we will run further experiments for these 3 settings. We want to find out how can we train even better model. In this testcase we'll increase the number of learning steps (ls) by 1000 up to ls = 5000 and compare the results with the data we already have, which is for ls = 1000. For the sake of brevity, results for Czech only will be stated here.

Data files: examplesCs_Ucca_All.txt
LRNN rules file: rulesExtended.txt

1000 - 5000 learning steps , extended LRNN rules set:

learning steps	train error	test error	test MSE	test AUC pr	test AUC roc
1000	0.198249	0.205625	0.151446	0.842747	0.703289
2000	0.198347	0.203625	0.155065	0.828153	0.688434
3000	0.198361	0.205375	0.153892	0.837192	0.702469
4000	0.198236	0.205625	0.151446	0.842753	0.703302
5000	0.198625	0.205375	0.153218	0.834405	0.702734

Table 5.13: Testcase 6a

LRNN rules file: rulesBasic.txt

1000 - 5000 learning steps , basic LRNN rules set:

learning steps	train error	test error	test MSE	test AUC pr	test AUC roc
1000	0.19825	0.206499	0.155815	0.829746	0.691603
2000	0.197916	0.204	0.152953	0.830635	0.695128
3000	0.198041	0.203874	0.151807	0.831099	0.696081
4000	0.198347	0.203499	0.152418	0.832958	0.700061
5000	0.198527	0.201874	0.152044	0.829786	0.691403

Table 5.14: Testcase 6b

LRNN rules file: rulesExtendedUcca.txt

1000 - 5000 learning steps , extended LRNN rules set
with UCCA annotations:

learning steps	train error	test error	test MSE	test AUC pr	test AUC roc
1000	0.194138	0.203125	0.174442	0.823884	0.683836
2000	0.198708	0.20325	0.155345	0.830382	0.6936
3000	0.198402	0.203125	0.155445	0.828313	0.689012
4000	0.199555	0.20625	0.153051	0.830787	0.694376
5000	0.198569	0.202749	0.151944	0.829519	0.695238

Table 5.15: Testcase 6c

When we look at the results, increasing the number of learning steps didn't fulfil the expectation, which was bigger difference between the train and test error. The AUC roc is best for the extended LRNN rules set without UCCA, so for the next testcase we will be working only with this rules set.

5.3.7 Testcase 7

Since the error values in the previous testcase didn't improve, in this testcase we will be working with the learning rate. The default value is 0.3, we will try to learn the model by using higher and lower learning rates.

Data files: examplesCs_Ucca_All.txt
LRNN rules file: rulesExtended.txt

learning rate	train error	test error	test MSE	test AUC pr	test AUC roc
0.1	0.198347	0.203625	0.150316	0.832914	0.699817
0.2	0.198222	0.20475	0.153599	0.831386	0.695958
0.3	0.198499	0.205499	0.156689	0.830022	0.692293
0.4	0.200555	0.205999	0.158978	0.858929	0.727563
0.5	0.224847	0.229499	9.604511E46	0.742039	0.480583
0.6	0.213874	0.210625	Infinity	0.804445	0.617375
0.7	0.231625	0.675	NaN	0.556213	8.205932E-4

Table 5.16: Testcase 7

The test error goes down with decreasing learning rate. The recommendation is to use lower learning rate rather than higher.

5.4 Discussion

In the practical part of this thesis two approach methods have been successfully tried and tested. The feature-based approach and the relational-based approach gave us clear results which can be compared to decide which approach has been more successful. Looking at the outcomes of our testcases, we can see, that the feature-based approach gives slightly lower error rates and bigger AUC values. The best results we obtained there was the last feature-based testcase, where additional semantic information feature was introduced. The improvement was not very big, yet still noticeable. The values of the AUC roc were always above 80% and sometimes reached even over 90%. This was never the case in the testcases for the relational-based approach. On the otherhand, relational-based results were not bad either and we were

able to find ways how to improve the learning process. As opposed to feature-based, rule-based approach didn't bring us much of an improvement when additional semantic information were introduced.

So the final conclusion is, that the recommended setting for the relational-based approach is the use of extended set of rules without UCCA annotations with the option of lr = 0.1. Increasing the number or learning steps didn't play any role, since the original ls = 1000 was already good enough. For the feature-based approach, the recommended setting is the one of the third testcase with UCCA annotations taken into account.

6. Conclusion

Natural Language Processing offers many areas for research, evaluation of the translation quality being one of them. In the theoretical part of this thesis, existing metrics for translation quality were discussed. Ways how to capture syntactic and semantic information were described. For our work we used some characteristics from different types of Machine Translation. Apart from the traditional ones which deal with syntax and morphology, we also used semantics to further improve our classifiers. Currently, human judgement on the translation is the decisive authority, computers have not fully replaced people yet, this however may change. The focus was on human evaluation metric which, together with tree representations were used to train neural networks.

There is still a lot of space left for future research. In particular, the use of semantic UCCA annotations may be further explored. In our experiments, we used only four out of twelve annotation types, there are another eight types which could be used as well. Also, UCCA is certainly not the only way how to capture semantics.

Another topic for future work is to enhance the feature-based representation. Initially, this approach ment to be only a benchmark for the relational-based approach, but since the results were quite good, this could be a good topic for future work.

7. Bibliography

[1] Abend O., Rappoport A. (2013)
Universal Conceptual Cognitive Annotation (UCCA) [online] http://www.aclweb.org/anthology/P13-1023

[2] Bahdanau D., Cho K., Bengio Y. (2016) *Neural machine translation by jointly learning to align and translate* [online] https://arxiv.org/abs/1409.0473

[3] Birch A. et al. (2016) *HUME: Human UCCA-Based Evaluation of Machine Translation* [online] https://arxiv.org/abs/1607.00030

[4] Bojar O. et al. (2016) *Results of the WMT16 Metrics Shared Tasks* [online] http://www.statmt.org/wmt16/pdf/W16-2302.pdf

[5] Bond F. (2006) *Machine Translation Introduction* [online] https://web.archive.org/web/20150914205051/http://www.csse.unimelb.edu.au/research/lt/nlp06/materials/Bond/mt-intro.pdf

[6] Cinková S. (2009) *From PropBank to EngValLex: Adapting the PropBank-Lexicon to the Valency Theory of the Functional Generative Description.* [online] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.380.2521

[7] Cinková S, et al. (2009)
Tectogrammatical Annotation of the Wall Street Journal [online]
https://ufal.mff.cuni.cz/pedt1.0/papers/cin_PEDT1_draft.pdf

[8] GIZA++: Training of statistical translation models. [online]
http://www.statmt.org/moses/giza/GIZA++.html

[9] Hutchins J. (2005) *The history of machine translation in a nutshell* [online] http://www.hutchinsweb.me.uk/Nutshell-2005.pdf

[10] Hssina B., et al. (2014)
A comparative study of decision tree ID3 and C4.5
[online] https://thesai.org/Downloads/SpecialIssueNo10/Paper_3-A_comparative_study_of_decision_tree_ID3_and_C4.5.pdf

[11] Lo C., Wu D. (2013) *Human Semantic MT Evaluation with HMEANT for IWSLT 2013* [online]
http://project.eu-bridge.eu/downloads/
Human_Semantic_MT_Evaluation_With_HMEANT_for_IWSLT2013.pdf

[12] Manning C.D., Raghavan P., Schütze H. (2008) *Introduction to Information Retrieval*

[13] Manning C.D., Schütze H. (2010) *Foundations of Statistical Natural Language Processing*

[14] Morris A.C. et al. (2004) *From WER and RIL to MER and WIL: improved evaluation measures for connected speech recognition* [online] https://www.researchgate.net/publication/
221478089_From_WER_and_RIL_to_MER_and_WIL_
improved_evaluation_measures_for_connected_speech_recognition

[15] Moses statistical machine translation system [online] http://www.statmt.org/moses

[16] Ney H., Essen U., Kneser R. (1995) *On the Estimation of 'Small' Probabilities by Leaving-One-Out*

[17] Och F.J., Ney H. (2000)
A comparison of alignment models for statistical machine learning [online] https://dl.acm.org/citation.cfm?id=992810

[18] Och F.J., Ney H. (2003)
A systematic comparison of various statistical alignment models [online] https://dl.acm.org/citation.cfm?id=778824

[19] Papineli K. et al. (2002)
BLEU: a Method for Automatic Evaluation of Machine Translation [online] http://aclweb.org/anthology/P/P02/P02-1040.pdf

[20] Quinlan J.R. (1993) *C4.5: Programs for machine learning*

[21] Serfozo R. (2009) *Basics of Applied Stochastic Processes*

[22] Šourek G. et al. (2015) *Lifted Relational Neural Networks* [online] https://arxiv.org/abs/1508.05128

[23] Uwents W. et al. (2011) *Neural networks for relational learning: an experimental comparison* [online] https://link.springer.com/article/10.1007/s10994-010-5196-5

[24] Wu Y. et al. (2016) *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* [online] https://arxiv.org/abs/1609.08144

8. Appendices

8.1 Appendix A - LRNN rules

case G..G:

```
a1(a) :- green(Y1), parent(X,Y1), green(Y2),
parent(X,Y2).
acc(X) :- green(Y1), parent(X,Y1), green(Y2),
parent(X,Y2).
0.0 finalKappa(a) :- a1(a).
```

case G:

```
a1(a) :- green(Y1), parent(X,Y1).
acc(X) :- green(Y1), parent(X,Y1).
0.0 finalKappa(a) :- a1(a).
```

case A G:

```
a1(a) :- acc(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
accx1(X) :- acc(Y1), parent(X,Y1), green(Y2),
```

```
parent(X,Y2).
...
...
accxn(X) :- [some rules]
acc(X) :- accx1(X).
...
...
acc(X) :- accxn(X).
0.0 finalKappa(a) :- a1(a).
```

case R..R:

```
a1(a) :- red(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
acc(X) :- red(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
0.0 finalKappa(a) :- a1(a).
```

case R:

```
a1(a) :- red(Y1), parent(X,Y1).
acc(X) :- red(Y1), parent(X,Y1).
0.0 finalKappa(a) :- a1(a).
```

case B R:

```
a1(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accx1(X) :- acc(Y1), parent(X,Y1), red(Y2),
parent(X,Y2).
...
...
```

```
accxn(X) :- [some rules]
acc(X) :- accx1(X).
...
...
acc(X) :- accxn(X).
0.0 finalKappa(a) :- a1(a).
```

case G O:

```
a1(a) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accx1(X) :- green(Y1), parent(X,Y1), acc(Y2),
parent(X,Y2).
...
accxn(X) :- [some rules]
acc(X) :- accx1(X).
...
acc(X) :- accxn(X).
0.0 finalKappa(a) :- a1(a).
```

case G R:

```
a1(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accx(X) :- green(Y1), parent(X,Y1), red(Y2),
parent(X,Y2).
0.0 finalKappa(a) :- a1(a).
```

extended UCCA rules:

```
a1(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
uccac(Y2).
a2(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
uccae(Y2).
a3(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
uccar(Y2).
a4(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
uccaf(Y2).
accx1(X) :- green(Y1), parent(X,Y1), red(Y2),
```

```
parent(X,Y2), uccac(Y2).
accx2(X) :- green(Y1), parent(X,Y1), red(Y2),
parent(X,Y2), uccae(Y2).
accx3(X) :- green(Y1), parent(X,Y1), red(Y2),
parent(X,Y2), uccar(Y2).
accx4(X) :- green(Y1), parent(X,Y1), red(Y2),
parent(X,Y2), uccaf(Y2).
acc(X) :- accx1(X).
acc(X) :- accx2(X).
acc(X) :- accx3(X).
acc(X) :- accx4(X).
0.0 finalKappa(a) :- a1(a).
0.0 finalKappa(a) :- a2(a).
0.0 finalKappa(a) :- a3(a).
0.0 finalKappa(a) :- a4(a).
```

case O R:

```
a1(a) :- orange(Y1), parent(X,Y1), red(Y2),
parent(X,Y2).
accx(X) :- orange(Y1), parent(X,Y1), red(Y2),
parent(X,Y2).
0.0 finalKappa(a) :- a1(a).
```

extended UCCA rules:

```
a1(a) :- orange(Y1), parent(X,Y1), red(Y2),
parent(X,Y2), uccac(Y2).
a2(a) :- orange(Y1), parent(X,Y1), red(Y2),
parent(X,Y2), uccae(Y2).
a3(a) :- orange(Y1), parent(X,Y1), red(Y2),
parent(X,Y2), uccar(Y2).
a4(a) :- orange(Y1), parent(X,Y1), red(Y2),
parent(X,Y2), uccaf(Y2).
accx1(X) :- orange(Y1), parent(X,Y1), red(Y2),
```

```
parent(X,Y2), uccac(Y2).
accx2(X) :- orange(Y1), parent(X,Y1), red(Y2),
parent(X,Y2), uccae(Y2).
accx3(X) :- orange(Y1), parent(X,Y1), red(Y2),
parent(X,Y2), uccar(Y2).
accx4(X) :- orange(Y1), parent(X,Y1), red(Y2),
parent(X,Y2), uccaf(Y2).
acc(X) :- accx1(X).
acc(X) :- accx2(X).
acc(X) :- accx3(X).
acc(X) :- accx4(X).
0.0 finalKappa(a) :- a1(a).
0.0 finalKappa(a) :- a2(a).
0.0 finalKappa(a) :- a3(a).
0.0 finalKappa(a) :- a4(a).
```

case G O R:

```
a1(a) :- green(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), red(Y3), parent(X,Y3).
accx(X) :- green(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), red(Y3), parent(X,Y3).
0.0 finalKappa(a) :- a1(a).
```

extended UCCA rules:

```
a1(a) :- green(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), red(Y3), parent(X,Y3), uccac(Y3).
a2(a) :- green(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), red(Y3), parent(X,Y3), uccae(Y3).
a3(a) :- green(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), red(Y3), parent(X,Y3), uccar(Y3).
a4(a) :- green(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), red(Y3), parent(X,Y3), uccaf(Y3).
accx1(X) :- green(Y1), parent(X,Y1), orange(Y2),
```

```
parent(X,Y2), red(Y3), parent(X,Y3), uccac(Y3).
accx2(X) :- green(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), red(Y3), parent(X,Y3), uccae(Y3).
accx3(X) :- green(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), red(Y3), parent(X,Y3), uccar(Y3).
accx4(X) :- green(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), red(Y3), parent(X,Y3), uccaf(Y3).
acc(X) :- accx1(X).
acc(X) :- accx2(X).
acc(X) :- accx3(X).
acc(X) :- accx4(X).
0.0 finalKappa(a) :- a1(a).
0.0 finalKappa(a) :- a2(a).
0.0 finalKappa(a) :- a3(a).
0.0 finalKappa(a) :- a4(a).
```

case A O R:

```
a1(a) :- acc(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), red(Y3), parent(X,Y3).
accx1(X) :- acc(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), red(Y3),
parent(X,Y3).
...
accxn(X) :- [some rules]
acc(X) :- accx1(X).
...
acc(X) :- accxn(X).
0.0 finalKappa(a) :- a1(a).
```

extended UCCA rules:

```
a1(a) :- acc(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), red(Y3), parent(X,Y3), uccac(Y3).
a2(a) :- acc(Y1), parent(X,Y1), orange(Y2),
```

```
parent(X,Y2), red(Y3), parent(X,Y3), uccae(Y3).
accx1(X) :- acc(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), red(Y3), parent(X,Y3), uccac(Y3).
accx2(X) :- acc(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), red(Y3), parent(X,Y3), uccae(Y3).
...
accxn(X) :- [some rules]
acc(X) :- accx1(X).
acc(X) :- accx2(X).
...
acc(X) :- accxn(X).
0.0 finalKappa(a) :- a1(a).
0.0 finalKappa(a) :- a2(a).
```

case A R:

```
a1(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accx1(X) :- acc(Y1), parent(X,Y1), red(Y2),
parent(X,Y2).
...
accxn(X) :- [some rules]
acc(X) :- accx1(X).
...
acc(X) :- accxn(X).
0.0 finalKappa(a) :- a1(a).
```

extended UCCA rules

```
a1(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
uccac(Y2).
a2(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2),
uccae(Y2).
accx1(X) :- acc(Y1), parent(X,Y1), red(Y2),
parent(X,Y2), uccac(Y2).
accx2(X) :- acc(Y1), parent(X,Y1), red(Y2),
```

```
parent(X,Y2), uccae(Y2).
...
accxn(X) :- [some rules]
acc(X) :- accx1(X).
acc(X) :- accx2(X).
...
acc(X) :- accxn(X).
0.0 finalKappa(a) :- a1(a).
0.0 finalKappa(a) :- a2(a).
```

case A O:

```
a1(a) :- acc(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2).
accx1(X) :- acc(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2).
...
accxn(X) :- [some rules]
acc(X) :- accx1(X).
...
acc(X) :- accxn(X).
0.0 finalKappa(a) :- a1(a).
```

case G O B:

```
a1(a) :- green(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), acc(Y3), parent(X,Y3).
accx1(X) :- green(Y1), parent(X,Y1), orange(Y2),
```

```
parent(X,Y2), acc(Y3), parent(X,Y3).
...
accxn(X) :- [some rules]
acc(X) :- accx1(X).
...
acc(X) :- accxn(X).
0.0 finalKappa(a) :- a1(a).
```

case O B:

```
a1(a) :- orange(Y1), parent(X,Y1), acc(Y2),
parent(X,Y2).
accx1(X) :- orange(Y1), parent(X,Y1), acc(Y2),
parent(X,Y2).
...
accxn(X) :- [some rules]
acc(X) :- accx1(X).
...
acc(X) :- accxn(X).
0.0 finalKappa(a) :- a1(a).
```

case G B:

```
a1(a) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accx1(a) :- green(Y1), parent(X,Y1), acc(Y2),
parent(X,Y2).
...
accxn(X) :- [some rules]
acc(X) :- accx1(X).
...
acc(X) :- accxn(X).
0.0 finalKappa(a) :- a1(a).
```

case A O B:

```
a1(a) :- acc(Y1), parent(X,Y1), orange(Y2),
parent(X,Y2), acc(Y3), parent(X,Y3).
accx1(a) :- acc(Y1), parent(X,Y1), orange(Y2),
```

```
parent(X,Y2), acc(Y3), parent(X,Y3).
...
accxn(X) :- [some rules]
acc(X) :- accx1(X).
...
acc(X) :- accxn(X).
0.0 finalKappa(a) :- a1(a).
```

8.2 Appendix B - Sets of LRNN rules for experiments

Generic rules

0.0 type1(X) :- red(X).
0.0 type1(X) :- green(X).
0.0 type1(X) :- orange(X).

0.0 type2(X) :- red(X).
0.0 type2(X) :- green(X).
0.0 type2(X) :- orange(X).

rule11(X) :- type1(Y), parent(X,Y), type1(Z), parent(X,Z).
rule12(X) :- type1(Y), parent(X,Y), type2(Z), parent(X,Z).
rule21(X) :- type2(Y), parent(X,Y), type1(Z), parent(X,Z).
rule22(X) :- type2(Y), parent(X,Y), type2(Z), parent(X,Z).

a_type1(X) :- type1(X).
a_type2(X) :- type2(X).

0.0 finalKappa(a) :- a_type1(X).
0.0 finalKappa(a) :- a_type2(X).

0.0 finalKappa(a) :- rule11(X).
0.0 finalKappa(a) :- rule12(X).
0.0 finalKappa(a) :- rule21(X).
0.0 finalKappa(a) :- rule22(X).

a) Set of Basic LRNN rules

aGG(a) :- green(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
aG(a) :- green(Y), parent(X,Y).
aAG(a) :- acc(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
accGG(X) :- green(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
accG(X) :- green(Y), parent(X,Y).

accAG(X) :- acc(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
aRR(a) :- red(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
aR(a) :- red(Y), parent(X,Y).
aBR(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accRR(X) :- red(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accR(X) :- red(Y), parent(X,Y).
accBR(X) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
aGO1(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
aGO2(a) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accGO1(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
accGO2(X) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
aGR(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accGR(X) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
aOR1(a) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
aOR2(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accOR1(X) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accOR2(X) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
orange(X) :- orange(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
orange(X) :- orange(Y), parent(X,Y).
acco1(X) :- orange(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
acco2(X) :- orange(Y), parent(X,Y).
0.0 acc(X) :- accGG(X).
0.0 acc(X) :- accG(X).
0.0 acc(X) :- accAG(X).
0.0 acc(X) :- accRR(X).
0.0 acc(X) :- accR(X).
0.0 acc(X) :- accBR(X).
0.0 acc(X) :- accGO1(X).
0.0 acc(X) :- accGO2(X).
0.0 acc(X) :- acco1(X).
0.0 acc(X) :- acco2(X).
0.0 acc(X) :- accGR(X).
0.0 acc(X) :- accOR1(X).
0.0 acc(X) :- accOR2(X).
0.0 finalKappa(a) :- aGG(a).

0.0 finalKappa(a) :- aG(a).
0.0 finalKappa(a) :- aAG(a).
0.0 finalKappa(a) :- aRR(a).
0.0 finalKappa(a) :- aR(a).
0.0 finalKappa(a) :- aBR(a).
0.0 finalKappa(a) :- aGO1(a).
0.0 finalKappa(a) :- aGO2(a).
0.0 finalKappa(a) :- aGR(a).
0.0 finalKappa(a) :- aOR1(a).
0.0 finalKappa(a) :- aOR1(a).

b) Set of Extended LRNN rules

aGG(a) :- green(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
aG(a) :- green(Y), parent(X,Y).
aAG(a) :- acc(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
accGG(X) :- green(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
accG(X) :- green(Y), parent(X,Y).
accAG(X) :- acc(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
orange(X) :- orange(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
orange(X) :- orange(Y), parent(X,Y).
acco1(X) :- orange(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
acco2(X) :- orange(Y), parent(X,Y).
aRR(a) :- red(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
aR(a) :- red(Y), parent(X,Y).
aBR(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accRR(X) :- red(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accR(X) :- red(Y), parent(X,Y).
accBR(X) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
aGO1(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
aGO2(a) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accGO1(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
accGO2(X) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
aGR(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accGR(X) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2).

aOR(a) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accOR(X) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
aGOR(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3).
accGOR(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3).
aAOR1(a) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3).
aAOR2(a) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), red(Y3), parent(X,Y3).
accAOR1(X) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3).
accAOR2(X) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), red(Y3), parent(X,Y3).
aAR(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accAR(X) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
aAO1(a) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
aAO2(a) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accAO1(X) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
accAO2(X) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
aGOB1(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), acc(Y3), parent(X,Y3).
aGOB2(a) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), acc(Y3), parent(X,Y3).
accGOB1(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), acc(Y3), parent(X,Y3).
accGOB2(X) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), acc(Y3), parent(X,Y3).
aOB1(a) :- orange(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
aOB2(a) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accOB1(X) :- orange(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accOB2(X) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
aGB(a) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accGB(X) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
aAOB1(a) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),

acc(Y3), parent(X,Y3).
aAOB2(a) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), acc(Y3), parent(X,Y3).
accAOB1(X) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), acc(Y3), parent(X,Y3).
accAOB2(X) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), acc(Y3), parent(X,Y3).
0.0 acc(X) :- accGG(X).
0.0 acc(X) :- accG(X).
0.0 acc(X) :- accAG(X).
0.0 acc(X) :- accRR(X).
0.0 acc(X) :- accR(X).
0.0 acc(X) :- accBR(X).
0.0 acc(X) :- accGO1(X).
0.0 acc(X) :- accGO2(X).
0.0 acc(X) :- acco1(X).
0.0 acc(X) :- acco2(X).
0.0 acc(X) :- accGR(X).
0.0 acc(X) :- accOR(X).
0.0 acc(X) :- accGOR(X).
0.0 acc(X) :- accAOR1(X).
0.0 acc(X) :- accAOR2(X).
0.0 acc(X) :- accAR(X).
0.0 acc(X) :- accAO1(X).
0.0 acc(X) :- accAO2(X).
0.0 acc(X) :- accGOB1(X).
0.0 acc(X) :- accGOB2(X).
0.0 acc(X) :- accOB1(X).
0.0 acc(X) :- accOB2(X).
0.0 acc(X) :- accGB(X).
0.0 acc(X) :- accAOB1(X).
0.0 acc(X) :- accAOB2(X).
0.0 finalKappa(a) :- aGG(a).
0.0 finalKappa(a) :- aG(a).
0.0 finalKappa(a) :- aAG(a).

0.0 finalKappa(a) :- aRR(a).
0.0 finalKappa(a) :- aR(a).
0.0 finalKappa(a) :- aBR(a).
0.0 finalKappa(a) :- aGO1(a).
0.0 finalKappa(a) :- aGO2(a).
0.0 finalKappa(a) :- aGR(a).
0.0 finalKappa(a) :- aOR(a).
0.0 finalKappa(a) :- aGOR(a).
0.0 finalKappa(a) :- aAOR1(a).
0.0 finalKappa(a) :- aAOR2(a).
0.0 finalKappa(a) :- aAR(a).
0.0 finalKappa(a) :- aAO1(a).
0.0 finalKappa(a) :- aAO2(a).
0.0 finalKappa(a) :- aGOB1(a).
0.0 finalKappa(a) :- aGOB2(a).
0.0 finalKappa(a) :- accOB1(a).
0.0 finalKappa(a) :- accOB2(a).
0.0 finalKappa(a) :- aGB(a).
0.0 finalKappa(a) :- aAOB1(a).
0.0 finalKappa(a) :- aAOB2(a).

c) Set of Extended LRNN rules with UCCA annotations

a1(a) :- green(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
aGG(a) :- green(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
aG(a) :- green(Y), parent(X,Y).
aAG(a) :- acc(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
accGG(X) :- green(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
accG(X) :- green(Y), parent(X,Y).
accAG(X) :- acc(Y1), parent(X,Y1), green(Y2), parent(X,Y2).
orange(X) :- orange(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
orange(X) :- orange(Y), parent(X,Y).
acco1(X) :- orange(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
acco2(X) :- orange(Y), parent(X,Y).

aRR(a) :- red(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
aR(a) :- red(Y), parent(X,Y).
aBR(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accRR(X) :- red(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accR(X) :- red(Y), parent(X,Y).
accBR(X) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
aGO1(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
aGO2(a) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accGO1(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
accGO2(X) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
aGR(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accGR(X) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
aGRuccac(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccac(Y2).
aGRuccae(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccae(Y2).
aGRuccar(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccar(Y2).
aGRuccaf(a) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccaf(Y2).
accGRuccac(X) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccac(Y2).
accGRuccae(X) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccae(Y2).
accGRuccar(X) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccar(Y2).
accGRuccaf(X) :- green(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccaf(Y2).
aOR(a) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accOR(X) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
aORuccac(a) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccac(Y2).
aORuccae(a) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccae(Y2).
aORuccar(a) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2),

uccar(Y2).
aORuccaf(a) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccaf(Y2).
accORuccac(X) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccac(Y2).
accORuccae(X) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccae(Y2).
accORuccar(X) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccar(Y2).
accORuccaf(X) :- orange(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccaf(Y2).
aGOR(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3).
aGORuccac(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccac(Y3).
aGORuccae(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccae(Y3).
aGORuccar(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccar(Y3).
aGORuccaf(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccaf(Y3).
accGOR(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3).
accGORuccac(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccac(Y3).
accGORuccae(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccae(Y3).
accGORuccar(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccar(Y3).
accGORuccaf(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccaf(Y3).
aAOR1(a) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3).
aAOR2(a) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), red(Y3), parent(X,Y3).

aAOR1uccac(a) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccac(Y3).
aAOR1uccae(a) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccae(Y3).
aAOR2uccac(a) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccac(Y3).
aAOR2uccae(a) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccae(Y3).
accAOR1(X) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3).
accAOR2(X) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), red(Y3), parent(X,Y3).
accAOR1uccac(X) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccac(Y3).
accAOR1uccae(X) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccae(Y3).
accAOR2uccac(X) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccac(Y3).
accAOR2uccae(X) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), red(Y3), parent(X,Y3), uccae(Y3).
aAR(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
aARuccac(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccac(Y2).
aARuccae(a) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccae(Y2).
accAR(X) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2).
accARuccac(X) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccac(Y2).
accARuccae(X) :- acc(Y1), parent(X,Y1), red(Y2), parent(X,Y2), uccae(Y2).
aAO1(a) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
aAO2(a) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accAO1(X) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2).
accAO2(X) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
aGOB1(a) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2),

acc(Y3), parent(X,Y3).
aGOB2(a) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), acc(Y3), parent(X,Y3).
accGOB1(X) :- green(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), acc(Y3), parent(X,Y3).
accGOB2(X) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), acc(Y3), parent(X,Y3).
aOB1(a) :- orange(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
aOB2(a) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accOB1(X) :- orange(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accOB2(X) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
aGB(a) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
accGB(X) :- green(Y1), parent(X,Y1), acc(Y2), parent(X,Y2).
aAOB1(a) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), acc(Y3), parent(X,Y3).
aAOB2(a) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), acc(Y3), parent(X,Y3).
accAOB1(X) :- acc(Y1), parent(X,Y1), orange(Y2), parent(X,Y2), acc(Y3), parent(X,Y3).
accAOB2(X) :- acc(Y1), parent(X,Y1), acc(Y2), parent(X,Y2), acc(Y3), parent(X,Y3).
0.0 acc(X) :- accGG(X).
0.0 acc(X) :- accG(X).
0.0 acc(X) :- accAG(X).
0.0 acc(X) :- accRR(X).
0.0 acc(X) :- accR(X).
0.0 acc(X) :- accBR(X).
0.0 acc(X) :- accGO1(X).
0.0 acc(X) :- accGO2(X).
0.0 acc(X) :- acco1(X).
0.0 acc(X) :- acco2(X).
0.0 acc(X) :- accGR(X).
0.0 acc(X) :- accGRuccac(X).
0.0 acc(X) :- accGRuccae(X).
0.0 acc(X) :- accGRuccar(X).

0.0 acc(X) :- accGRuccaf(X).
0.0 acc(X) :- accOR(X).
0.0 acc(X) :- accORuccac(X).
0.0 acc(X) :- accORuccae(X).
0.0 acc(X) :- accORuccar(X).
0.0 acc(X) :- accORuccaf(X).
0.0 acc(X) :- accGOR(X).
0.0 acc(X) :- accGORuccac(X).
0.0 acc(X) :- accGORuccae(X).
0.0 acc(X) :- accGORuccar(X).
0.0 acc(X) :- accGORuccaf(X).
0.0 acc(X) :- accAOR1(X).
0.0 acc(X) :- accAOR2(X).
0.0 acc(X) :- accAOR1uccac(X).
0.0 acc(X) :- accAOR1uccae(X).
0.0 acc(X) :- accAOR2uccac(X).
0.0 acc(X) :- accAOR2uccae(X).
0.0 acc(X) :- accAR(X).
0.0 acc(X) :- accAO1(X).
0.0 acc(X) :- accAO2(X).
0.0 acc(X) :- accGOB1(X).
0.0 acc(X) :- accGOB2(X).
0.0 acc(X) :- accOB1(X).
0.0 acc(X) :- accOB2(X).
0.0 acc(X) :- accGB(X).
0.0 acc(X) :- accAOB1(X).
0.0 acc(X) :- accAOB2(X).
0.0 finalKappa(a) :- aGG(a).
0.0 finalKappa(a) :- aG(a).
0.0 finalKappa(a) :- aAG(a).
0.0 finalKappa(a) :- aRR(a).
0.0 finalKappa(a) :- aR(a).
0.0 finalKappa(a) :- aBR(a).
0.0 finalKappa(a) :- aGO1(a).
0.0 finalKappa(a) :- aGO2(a).

```
0.0 finalKappa(a) :- aGR(a).
0.0 finalKappa(a) :- aGRuccac(a).
0.0 finalKappa(a) :- aGRuccae(a).
0.0 finalKappa(a) :- aGRuccar(a).
0.0 finalKappa(a) :- aGRuccaf(a).
0.0 finalKappa(a) :- aOR(a).
0.0 finalKappa(a) :- aORuccac(a).
0.0 finalKappa(a) :- aORuccae(a).
0.0 finalKappa(a) :- aORuccar(a).
0.0 finalKappa(a) :- aORuccaf(a).
0.0 finalKappa(a) :- aGOR(a).
0.0 finalKappa(a) :- aGORuccac(a).
0.0 finalKappa(a) :- aGORuccae(a).
0.0 finalKappa(a) :- aGORuccar(a).
0.0 finalKappa(a) :- aGORuccaf(a).
0.0 finalKappa(a) :- aAOR1(a).
0.0 finalKappa(a) :- aAOR2(a).
0.0 finalKappa(a) :- aAOR1uccac(a).
0.0 finalKappa(a) :- aAOR1uccae(a).
0.0 finalKappa(a) :- aAOR2uccac(a).
0.0 finalKappa(a) :- aAOR2uccae(a).
0.0 finalKappa(a) :- aAR(a).
0.0 finalKappa(a) :- aAO1(a).
0.0 finalKappa(a) :- aAO2(a).
0.0 finalKappa(a) :- aGOB1(a).
0.0 finalKappa(a) :- aGOB2(a).
0.0 finalKappa(a) :- accOB1(a).
0.0 finalKappa(a) :- accOB2(a).
0.0 finalKappa(a) :- aGB(a).
0.0 finalKappa(a) :- aAOB1(a).
0.0 finalKappa(a) :- aAOB2(a).
```

9. Content of CD

- inputData: input data in two .xlsx files

- LRNNDemo: demo examples used in section 4.2

- LRNNInputData: input data for relational-based testcases

- LRNNResults: results of all relational-based testcases

- LRNNRules: rules as described in the section 4.3

- parser: java code of the parser

- tex: tex code of this thesis

- WEKADemo: demo examples used in section 3.2

- WEKAInputData: input data for feature-based testcases

- WEKAResults: results of all feature-based testcases

- infoLRNN.txt: example of LRNN java command

- thesis.pdf: pdf file with this thesis