CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering

Department of Computer Science

Master's Thesis

Automatic evaluation of word problems

Bc. Jan Kadlec

Open Informatics, Data Science

May 2022

Supervisor: doc. RNDr. Daniel Průša, Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kadlec** Jméno: **Jan** Osobní číslo: **474727**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Datové vědy**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Automatické vyhodnocování slovních úloh

Název diplomové práce anglicky:

Automatic evaluation of word problems

Pokyny pro vypracování:

The aim of the thesis is to propose and implement a method for automatic evaluation of mathematical word problems in Czech at the level of 1st to 4th grade of elementary school.

Detailed instructions:

- Get acquainted with existing methods of solving word problems in English, explain their principles.
- Base your solution on deep learning.
- Consider only word problems that can be solved by evaluating an arithmetic expression.
- Use the training and test data prepared during the software project, or expand this data set accordingly.
- Evaluate the implemented method empirically. Provide a thorough analysis of the achieved results. Compare the solution with state-of-the-art methods (at least indirectly).

Seznam doporučené literatury:

[1] D. Zhang, L. Wang, L. Zhang, B. T. Dai, H. T. Shen: The Gap of Semantic Parsing: A Survey on Automatic Math Word Problem Solvers, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 9, pp. 2287-2305, 2020.

[2] I. Goodfellow, Y. Bengio, A. Courville: Deep Learning, The MIT Press, 2016.

[3] J. Kadlec, D. Průša: Solvers for Mathematical Word Problems in Czech, Proceedings of the 20th Conference Information Technologies - Applications and Theory, 2020.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. RNDr. Daniel Průša, Ph.D. Strojové učení FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **02.02.2022** Termín odevzdání diplomové práce: **20.05.2022**

Platnost zadání diplomové práce: **30.09.2023**

doc. RNDr. Daniel Průša, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.

Datum převzetí zadání Podpis studenta

Acknowledgement / Declaration

I would like to thank my supervisor, doc. RNDr. Daniel Průša, Ph.D., for his professional guidance on my thesis.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 20. 5. 2022

....

Abstrakt / Abstract

Tato diplomová práce shrnuje vývoj a poslední trendy solverů pro slovní úlohy - jedná se o starou úlohu zpracování přirozeného jazyka, která se datuje do 60. let minulého století. Tato práce se zaměřuje na historii solverů, vytváření datasetu, sequence-to-sequence (seq2seq) modelu pro strojový překlad a implementaci solveru, založeném na seq2seq modelu pro slovní úlohy v českém jazyce a ověření na vytvořeném datasetu. Nakonec se diskutuje o budoucnosti solverů pro slovní úlohy a jejich použití.

Klíčová slova: zpracování přirozeného jazyka, strojové učení, automatické vyhodnocení, rekurentní neuronové sítě, slovní úloha, český jazyk

Překlad titulu: Automatické vyhodnocování slovních úloh

This diploma thesis summarizes the evolution and trends in word problem solvers - an old natural language processing (NLP) task whose foundations date back to the 1960s. This work focuses on the history of solvers, dataset creation, sequence-to-sequence (seq2seq) model for machine translation, and solver implementation based on the seq2seq model for word problems in Czech and its evaluation on the created dataset. Finally, the future of the word problem solvers and their application will be discussed.

Keywords: natural language processing, machine learning, automatic evaluation, recurrent neural networks, word problem, Czech language

/ Contents

Tables / Figures

Chapter 1
Introduction

Automatic evaluation of word problems is an old NLP task, which dates back to 1964, but is still actual and, thanks to the new machine learning approaches, the methods of solving this problem are still improving. The input to this task is a mathematical word problem formulated in a natural language, and the output is the result, the equation(s) or a procedure to obtain the result. This task may look trivial by its description, but the opposite is true. It is similar to natural language translation: we are looking for a corresponding *translation* of a word problem to a desired output with a substantive difference that a slight change in the input text can result in a major change in the output.

1.1 History

The history of word problem solvers is best described in [1]. This work identifies three areas based on the period and used techniques. Let us take a closer look at these approaches and discuss techniques and related solvers.

The pioneer among the word problem solvers is considered to be a solver STUDENT by Daniel G. Bobrow, presented in his work [2] in 1964. The solver is written in the Lisp programming language. It accepts word problems in English and outputs the answer to the word problem (text answer with a number). First, the solver substitutes some selected tokens (i.e., twice becomes 2 times). Then the words in the word problem are tagged using a dictionary. The following part breaks the word problem into smaller parts, called simple sentences. After this step, the transformation to mathematical expressions is applied. A year later, the word problem solver DEDUCOM [3] was presented. DEDUCOM is just like STUDENT written in the Lisp programming language. It uses 68 facts, representing general knowledge, and based on them, deduces the answer to a question. Another well-known word problem solver is WORDPRO presented in 1984 by Charles R. Fletcher [4]. Unlike STUDENT and DEDUCOM, WORDPRO is written in an Interlisp-D programming environment built around Lisp. The solver works on a third-grade student's restricted set of word problems. The algorithm does transformation, and after that, it applies rules, which yields the result. These all mentioned solvers have several things in common. All of them are written in the Lisp programming language or in the Lisp-based programming environment. They all use rule-based matching, using a transformation of input and then matching the rules. Last but not least, it is hard for others to compare which solver is better because there was no public testing dataset back then.

We can consider the second era of word problem solvers as a foundation for solvers as we know them today. This era brought us standardisation measures for word problem solvers, well-established datasets. The one we cannot forget to mention is the Alg514 dataset. It consists of 514 word problems in English and was presented together with a word problem solver in [5] by Kushman et al. in 2014. Since then, other datasets have been published, but Alg514 is considered to be the first standardised dataset. Except for Alg514, Kushman et al. presented us with a solver in their work from 2014. The solver first selects a template. The templates are extracted from the train set, which means that the solver can solve word problems that have the same templates as word problems from the train set. The template is selected using a probabilistic model. Then, the numbers and variables from the word problem are mapped to the selected template, resulting in a so-called aligned template. After these steps, instantiated equations are extracted and solved, producing the result. It achieves 69% accuracy on Alg514. The solver is related to two other research topics: situated semantic interpretation and information extraction. Unfortunately, the dataset Alg514 lacks scale, as 514 word problem instances are not enough for training and testing a general, robust solver. This is why there was a need for more extensive datasets. Huang et al. took on this task, and in their work [6] they presented the Dolphin18K dataset, which contains more than 18,000 annotated word problems in English. In Chapter 3, we will discuss the quality of available datasets. Dolphin18K dataset was used to train and test the solvers existing at that time and showed that they cannot cope with a large-scale dataset with various word problems. The second era of word problem solvers brought us standardisation of datasets, metrics for comparing solvers accuracy. Solvers of this era are based on semantic parsing, feature engineering, and statistical learning.

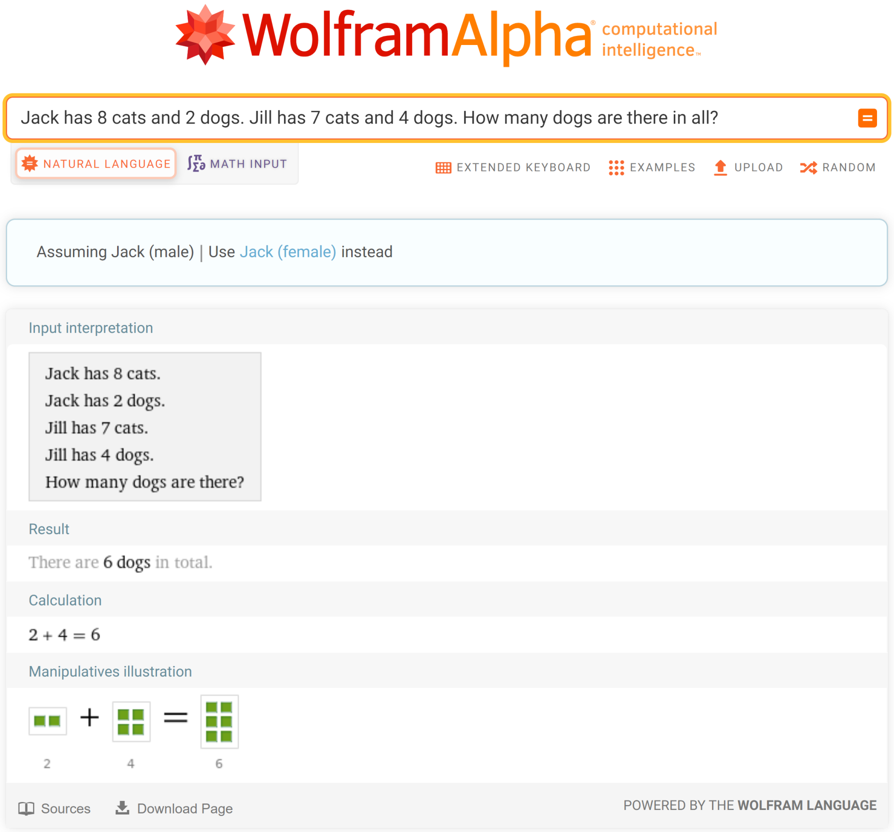
The latest solvers from the third era are based on deep learning and reinforcement learning. A significant change occurred with the seq2seq model and other NLP-based methods. This era brought many significant benefits for word problem solvers and not just for its success in solving word problems, but mainly due to datasets it brought and still brings. DNS is the first solver from this era, based purely on deep learning methods, meaning that the related algorithms do not use hand-crafted features. DNS was presented by Wang et al. [7] in 2017 and uses the seq2seq model. The seq2seq model was developed for machine translation and was introduced in 2014 [8], so in 2017, it was still a new technology when DNS was introduced. A year after DNS publication, another solver was proposed - Seq2SeqET. The Seq2SeqET solver [9] is an extension to DNS, which brings several modifications to improve accuracy. Improvements include equation normalisation, BiLSTM, ConvS2S, Transformer, and Ensemble. We should pay particular attention to these improvements because the work shows that, thanks to these improvements, the accuracy increased from 58.1% (DNS) to 68.4% (DNS with improvements) when tested on dataset Math23K [7]. Last but not least, there is the MathDQN solver [10], based on a reinforcement learning framework called deep Q-network. It achieves accuracy of 76.0% on ArithM dataset [10]. These solvers use the latest machine learning approaches to achieve the best possible accuracy, and as we can see in the results, they are doing great.

1.2 Existing online solvers

There are few word problem solvers that are available online. Let us mention two of them: WolframAlpha[1] and Symbolab[2]. These solvers are examples of the ultimate goal of word problem solvers, an interface for typing word problems and pressing the “solve” button to uncover the step-by-step solution. Unfortunately, they have poor accuracy,

[1] https://www.wolframalpha.com/

[2] https://www.symbolab.com/



Jack has 8 cats and 2 dogs. Jill has 7 cats and 4 dogs. How many dogs are there in all?

Assuming Jack (male) | Use Jack (female) instead

Input interpretation

Jack has 8 cats.
Jack has 2 dogs.
Jill has 7 cats.
Jill has 4 dogs.
How many dogs are there?

Result

There are 6 dogs in total.

Calculation

$2 + 4 = 6$

Manipulatives illustration

2 + 4 = 6

Figure 1.1. WolframAlpha solver applied to solve the word problem “Jack has 8 cats and 2 dogs. Jill has 7 cats and 4 dogs. How many dogs are there in all?”.

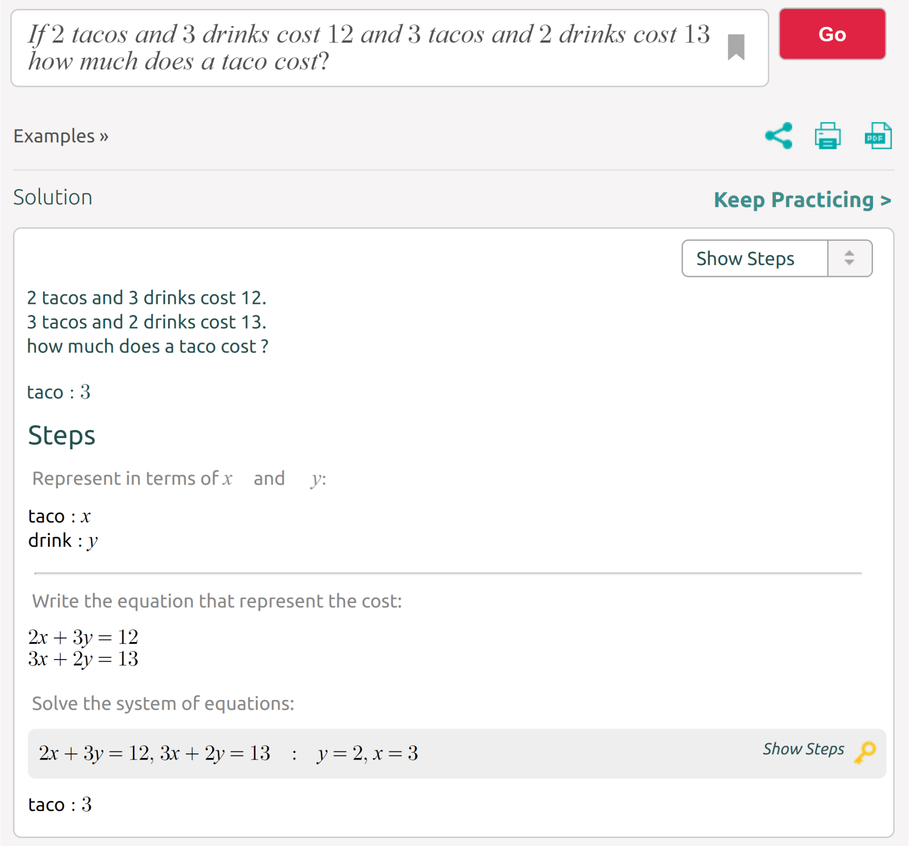
and after examination, we can conclude that they are template-based. Let us look at the examples in Figure 1.1 and Figure 1.2.

It is essential to note that both solvers are focused on different word problems. WolframAlpha extracts information from the word problem and answers the question with its knowledge. Symbolab, on the other hand, creates a system of equation(s) and answers the question of the word problem by solving the system of equation(s). As mentioned before, both solvers are probably template-based, meaning that we can easily confuse the solver to be unable to solve a word problem, as demonstrated in Figure 1.3. Let us have a closer look at this issue.

1.3 State-of-the-art

It is challenging to identify a state-of-the-art word problem solver. Recently, there have been a lot of new publications describing new word problem solvers and presenting breaking boosts in their accuracy, but unfortunately, we can not consider a lot of them as reliable because they do not provide a specific implementation of their solvers. Another reason why it is hard to identify a state-of-the-art solver is that this diploma thesis focuses on word problems in Czech, and there are not many solvers and even datasets for this type of word problem.

My bachelor thesis[11] focused on word problems in Czech and proposed two solvers based on solvers from the second era. This thesis will consider a state-of-the-art word



If 2 tacos and 3 drinks cost 12 and 3 tacos and 2 drinks cost 13 how much does a taco cost?

2 tacos and 3 drinks cost 12.
3 tacos and 2 drinks cost 13.
how much does a taco cost ?

taco : 3

Steps

Represent in terms of x and y:

taco : x
drink : y

Write the equation that represent the cost:

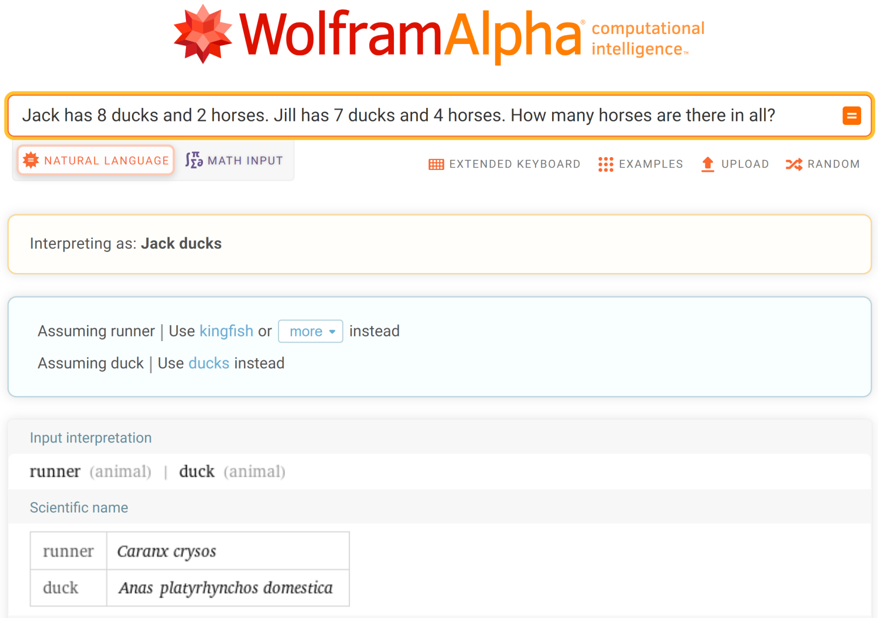
$2x+3y=12$
$3x+2y=13$

Solve the system of equations:

$2x+3y=12, 3x+2y=13 \quad : \quad y=2, x=3$ Show Steps

taco : 3

Figure 1.2. Symbolab solver applied to solve the word problem "If 2 tacos and 3 drinks cost 12 and 3 tacos and 2 drinks cost 13 how much does a taco cost?".



Jack has 8 ducks and 2 horses. Jill has 7 ducks and 4 horses. How many horses are there in all?

Interpreting as: **Jack ducks**

Assuming runner | Use kingfish or more instead
Assuming duck | Use ducks instead

Input interpretation

runner (animal) | duck (animal)

Scientific name

runner	*Caranx crysos*
duck	*Anas platyrhynchos domestica*

Figure 1.3. Slightly modifying an example word problem results in the solver being unable to solve it correctly. Modified word problem "Jack has 8 ducks and 2 horses. Jill has 7 ducks and 4 horses. How many horses are there in all?".

problem solver of the latest era, DNS. Although there is no source code, the implementation is straightforward. The majority of the latest solvers are based on this approach. For example, there are two solvers [12] [13] from 2021 that directly use the foundation concept of the DNS - seq2seq model or partly use it.

1.4 Problem definition

This thesis aims to propose and implement a method to automatically evaluate mathematical word problems in Czech at the level of 1st to 4th grade of elementary school. For a further specification of word problems, we will consider only word problems that are solved by a single equation. This requirement is based on known solvers, where some solvers focus on the specific type of word problems, which results in better proposed solutions. The solver is based on the latest NLP and machine learning approaches. The output of the solver can differ. We can evaluate the solver accuracy by the numeric result or compare the equation returned by the solver. Let us have a further insight into these measures in the chapter that deals with the experiment.

Let us look at an example of a word problem that satisfies the requirements.

```
Na parkovišti stály 4 autobusy a 9 krát více aut.
Kolik dopravních prostředků bylo celkem na parkovišti?
```

The word problem translation to English is:

```
There were 4 buses and 9 times more cars in the parking lot.
How many vehicles were in the parking lot in total?
```

The word problem above is solved by the following equation.

```
4 + 4 * 9
```

The result of the word problem is 40.

Chapter 2
Related topics

This chapter will look at related topics of the proposed word problem solvers, dataset and sequence to sequence. We will discuss existing datasets and the creation of the Czech word problems dataset, which the proposed solver will use. Afterwards, we will introduce the sequence-to-sequence approach and mention its principle, different forms, and extensions.

2.1 Dataset

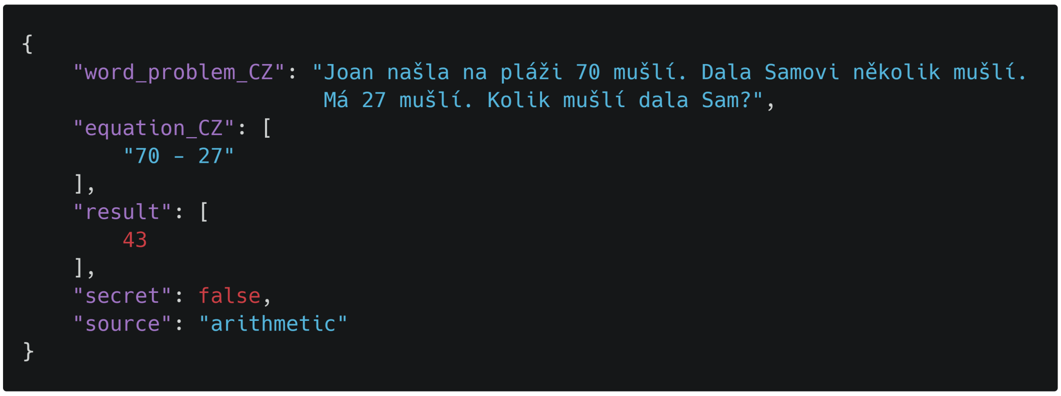
As mentioned in Subchapter 1.4, this thesis aims to evaluate mathematical word problems in Czech, which means that a dataset of the Czech word problem dataset is required. My bachelor thesis[11] introduced a dataset of word problems in Czech. This dataset consists of 499 word problems, which is unfortunately not sufficient for a general solver. For this purpose, a larger dataset of Czech word problems was created. Let us describe the used dataset creation process before we have a look at the new dataset.

Creating a dataset is not an effortless task and is crucial for future work and evaluation itself. One of the first things that come to mind when creating a dataset is what storage to use. We can find inspiration in existing datasets that use schemaless data storage, JSON. The vast advantages of using JSON as storage are easy to access, editing, extending, and accessing using several popular technologies, such as Python and MongoDB. Even though JSON is a schemaless data storage, we would like our storage to follow a schema that we will know.

The created dataset has the following schema:

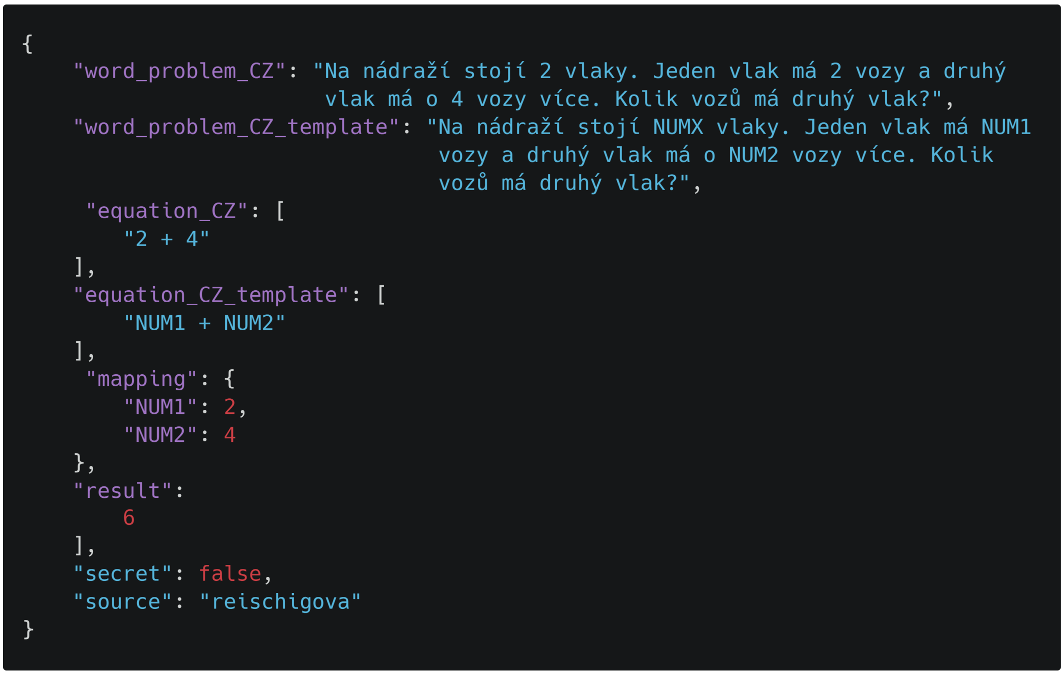
attribute	data type	optional (empty means False)
word_problem_CZ	string	
equation_CZ	list	
result	list	
source	string	
secret	bool	
word_problem_CZ_template	string	True
equation_CZ_template	list	True
mapping	list	True
word_problem_original	string	True
equation_original	list	True
units	list	True

Table 2.1. Schema for dataset.



```
{
    "word_problem_CZ": "Joan našla na pláži 70 mušlí. Dala Samovi několik mušlí.
                        Má 27 mušlí. Kolik mušlí dala Sam?",
    "equation_CZ": [
        "70 - 27"
    ],
    "result": [
        43
    ],
    "secret": false,
    "source": "arithmetic"
}
```

Figure 2.1. An example of a word problem record with only mandatory attributes.



```
{
    "word_problem_CZ": "Na nádraží stojí 2 vlaky. Jeden vlak má 2 vozy a druhý
                        vlak má o 4 vozy více. Kolik vozů má druhý vlak?",
    "word_problem_CZ_template": "Na nádraží stojí NUMX vlaky. Jeden vlak má NUM1
                                 vozy a druhý vlak má o NUM2 vozy více. Kolik
                                 vozů má druhý vlak?",
     "equation_CZ": [
        "2 + 4"
    ],
    "equation_CZ_template": [
        "NUM1 + NUM2"
    ],
     "mapping": {
        "NUM1": 2,
        "NUM2": 4
    },
    "result":
        6
    ],
    "secret": false,
    "source": "reischigova"
}
```

Figure 2.2. An example of a word problem record with optional fields *word_problem_CZ_template*, *equation_CZ_template* and *mapping* due to not deterministic mapping between word problem and equation.

The attributes *word_problem_CZ* and *equation_CZ* are the most important ones for the model because they contain information about the word problem and the equation that solves it. When the direct mapping of numbers between *word_problem_CZ* and *equation_CZ* is not deterministic, attributes *word_problem_CZ_template*, *equation_CZ_template* and *mapping* are used. It should be pointed out that *word_problem_CZ* and *equation_CZ* follow some rules. These rules establish data standardisation and consistency, which is crucial for a dataset. We want the attribute *word_problem_CZ* to follow the Czech grammar rules. Unfortunately, as we will see on the created dataset, this demand does not have to be fully satisfied for some sets of word problems because of translation to the Czech language.

The rules for the attribute *equation_CZ* are the following:

- the allowed maths operation symbols are $+$, $-$, $*$, $/$

- parentheses (and) are allowed
- before the maths operation symbols and after them there is space

An example of *equation_CZ* value following standard:

```
(3 + 1) * (5 - 4) + 10 / 5
```

As mentioned in Subchapter 1.4, let us remind others of the standardisation we want our word problems to follow.

- the word problem is solved using a single equation (of one unknown variable)
- the result of the word problem is an integer
- there exists a mapping between the assignment of the word problem and all the numbers that appear in the equation (no hidden or additional knowledge is required to solve it);

The reason to consider only word problems solved using an equation of one unknown variable was mentioned before. If we consider the results to have noninteger values, we could face problems with simple arithmetic operations because computers generally struggle with floating points[1].

To create the dataset, I used the translation of existing word problem datasets in foreign languages into Czech language. Some of the datasets can be seen in Figure 2.2. It should be mentioned that finding a dataset from a paper is a challenging task. Sometimes there is a link in the paper, but usually it is a dead link leading to a HTTP 404 error. Other sources of word problems are schoolbooks or collections of math word problems. Unfortunately, the number of these sources is limited. The schoolbooks and collections used to create datasets can be seen in Figure 2.3. The Internet is the third source where word problems were found. It is worth mentioning that extracting information from some web pages is challenging. All used web pages can be seen in Figure 2.4.

dataset	# word problems	language	year	source
Alg514	514	English	2014	[5]
AllArith*	831	English	2017	[14]
Ape210K*	210,488	Chinese	2020	[15]
AQuA	100,000	English	2017	[16]
Dolphin18K	18,460	English	2016	[6]
Math23K	23,161	Chinese	2017	[7]
MAWPS	3,320	English	2016	[17]

Table 2.2. Existing datasets. Datasets with * were translated to the Czech and processed.

Unfortunately, not all dataset word problems are suitable for the resulting dataset, as they do not follow the rules set during dataset creation. That is why word problems have to be filtered and standardised. The final number of word problems in the created dataset is 23,568. The distribution of the eight most common equations in the dataset can be observed in Figure 2.3. We can find more accurate statistics of the equations in Appendix B.1.

[1] In *Python 3.8.10* equation $1.2 - 1$ results is 0.19999999999999996

schoolbook/collection	# of used word problems	source
Matematika na základní a obecné škole ve slovních úlohách	499	[18]
SLOVNÍ ÚLOHY PRO 2.-5. ROČNÍK	303	[19] [20] [21] [22]

Table 2.3. Schoolbooks and collections that were processed. Word problems from *Matematika na základní a obecné škole ve slovních úlohách* is known from my bachelor thesis as *WP500* dataset. Unfortunately, we found a duplicate word problem, so in this thesis, we will not call this subset of word problem as *WP500*.

link	# of used word problems
`https://www.umimematiku.cz/`	612
`https://www.hackmath.net/`	67
`https://www.matika.in/`	25

Table 2.4. Internet sources that were processed.

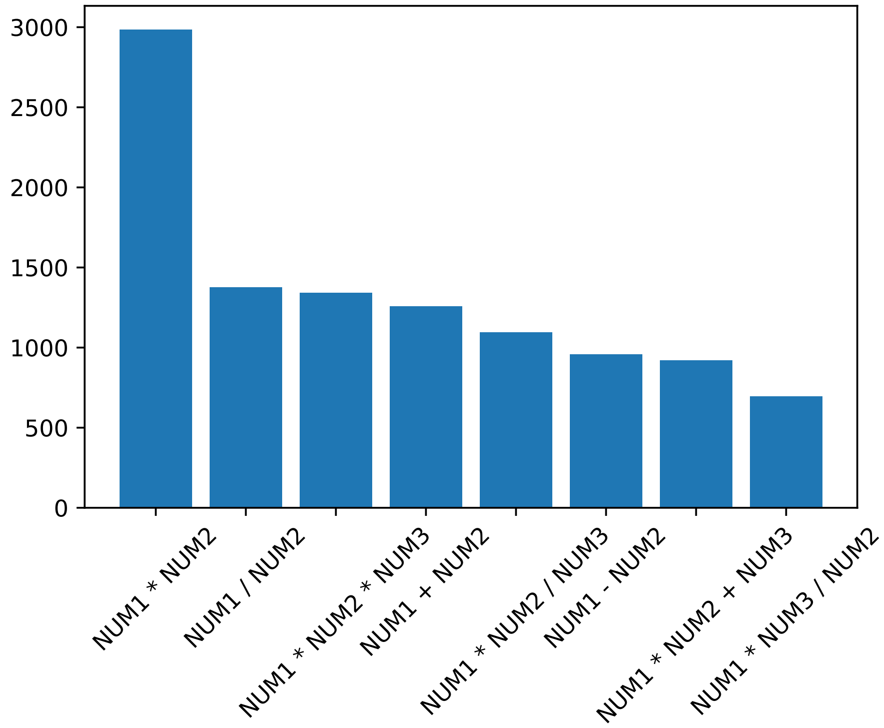


Figure 2.3. The figure shows the eight most common equation templates from the dataset and the number of their occurrences.

2.2 Sequence to sequence

As mentioned before, the seq2seq model was presented in 2014 by a group of researchers from Google[8]. The motivation to create sequence-to-sequence learning was the lack of deep neural networks (DNNs) to map an input sequence to an output sequence, which is, in fact, the case of machine translation. The cited paper experiments with translation from English to French and vice versa, outperforming existing state-of-the-art. Years later, Google presented an open source sequence-to-sequence framework in Tensor Flow, which we can still find on GitHub[2]. Also, Google started using a sequence-to-sequence model called Google Neural Machine Translation [23] in its well-known production systems, such as Google Translator[3].

2.2.1 Principle

As is evident from the name, seq2seq performs the mapping from one sequence to another, and it became successful in machine translation. The formal definition for machine translation is that we have a sequence $x_1, x_2, ..., x_m$ (tokens, where x_i is a token of input sequence on position i) and another sequence $y_1, y_2, ..., y_n$ (tokens, where y_i is a token of output sequence on position i), and the task is to find the most probable output sequence by the given input sequence. Note that both sequences do not have to be of the same length.

The model consists of two primary components, the encoder and the decoder, where each is, in fact, a recurrent neural networks (RNNs). The most popular ones are Long short-term memory (LSTM) and Gated recurrent unit (GRU). Both will be discussed in more detail in the thesis.

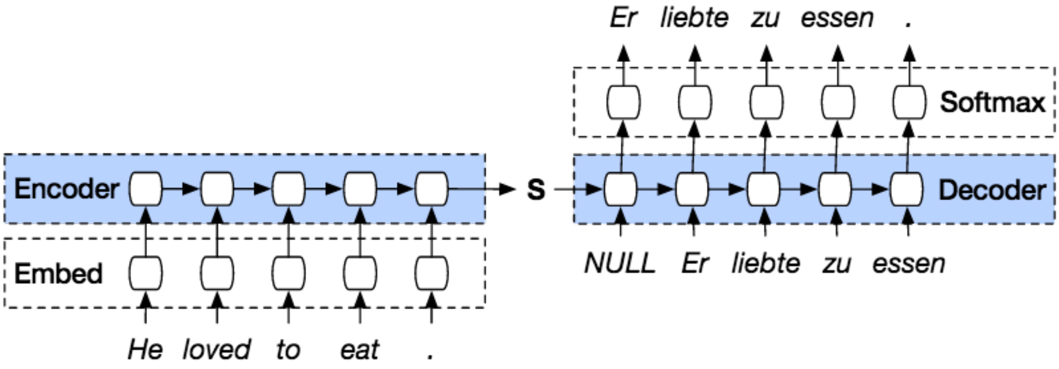


Figure 2.4. An example of seq2seq model[24] used for translation from English to German.

LSTM is a recurrent neural network architecture commonly used in NLP tasks. The architecture was presented in 1997 by Sepp Hochreiter and Jürgen Schmidhuber [25]. The neural network consists of several units: cell, input gate, output gate, and forget gate. There are several variants of the LSTM architecture. In this thesis, we will work with a variant that PyTorch[4] uses, which is based on the work from 2014 of the Google research group [26].

LSTM computes following:

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi})$$

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf})$$

$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg})$$

[2] `https://github.com/google/seq2seq`

[3] `https://translate.google.com/`

[4] `https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html`

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho})$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

$$h_t = o_t \odot \tanh(c_t)$$

Where, for a time t,

x_t is the input

h_t is the hidden state

i_t is a computation of the input gate

f_t is a computation of the forget gate

g_t is a computation of the cell

o_t is a calculation of the output gate

c_t is the cell state

σ is the sigmoid function

$\odot$ is the Hadamard product

W is the weight matrix

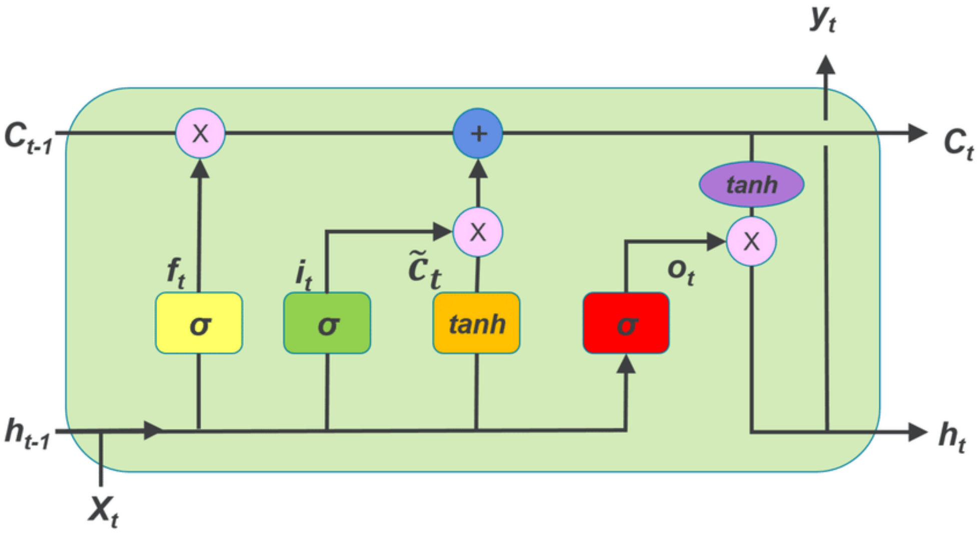


Figure 2.5. Schema for LSTM architecture from [27]. Note that symbol "X" represents $\odot$ — Hadamard product.

A GRU is another popular recurrent neural network used in the seq2seq model. GRU was presented in 2014 by Kyunghyun Cho et al. [28]. The most significant difference between GRU and LSTM is the number of parameters. GRU has fewer parameters than LSTM, caused by the lack of an output gate in GRU, and it is worth mentioning that the names of GRU units are different. Instead of the input, forget gate, GRU has reset, update, and new gates. As we examined the variant of LSTM using PyTorch, we will do the same for GRU [5].

[5] https://pytorch.org/docs/stable/generated/torch.nn.GRU.html

GRU computes following:

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr})$$

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz})$$

$$n_t = tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{(t-1)} + b_{hn}))$$

$$h_t = (1 - z_t) \odot n_t + z_t \odot h_{(t-1)}$$

Where, for a time t,

x_t is the input

h_t is the hidden state

r_t is a computation of the reset gate

z_t is a computation of the update gate

n_t is a computation of the new gate

σ is the sigmoid function

$\odot$ is the Hadamard product

W is the weight matrix

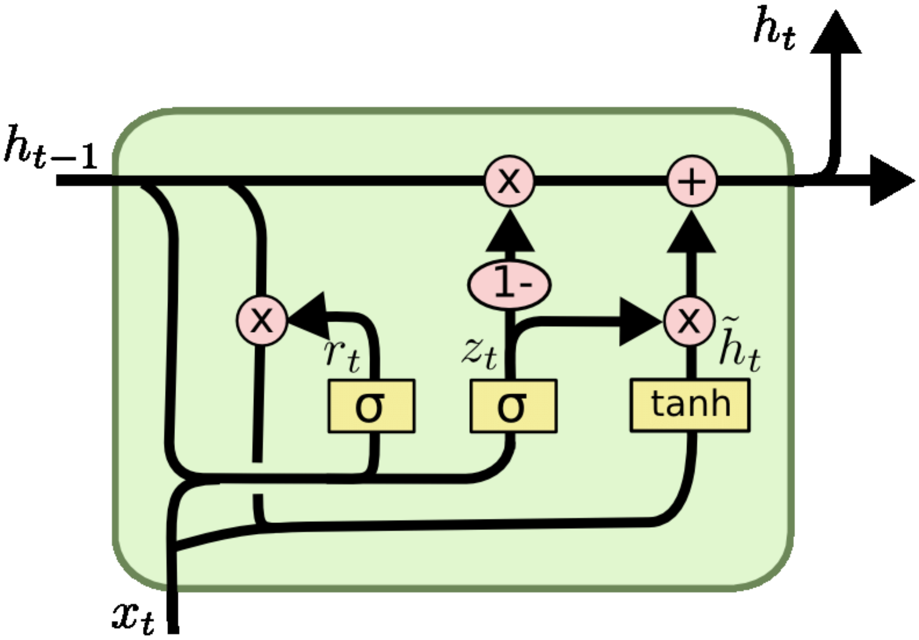


Figure 2.6. Schema for GRU architecture from [29]. Note that symbol "X" represents $\odot$ — Hadamard product and symbol $\tilde{h}_t$ is in fact n_t.

In summary, for the LSTM and GRU architectures, GRU has fewer parameters, which means that we can expect faster learning and a smaller size of the trained model. On the other hand, we will probably tend to use LSTM in large datasets due to the number of parameters and complexity, resulting in better performance. The reason why LSTM and GRU are used is to avoid the problem of vanishing gradients.

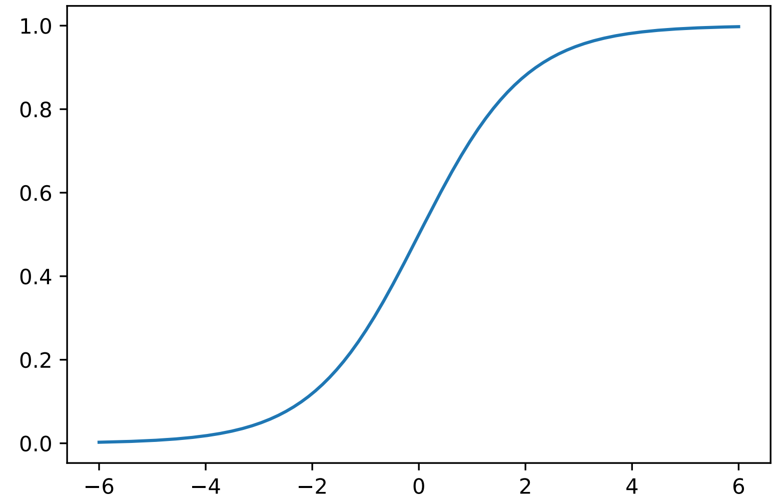


Figure 2.7. Figure of sigmoid function

$$S(x) = \frac{1}{1+e^{-x}}$$

.

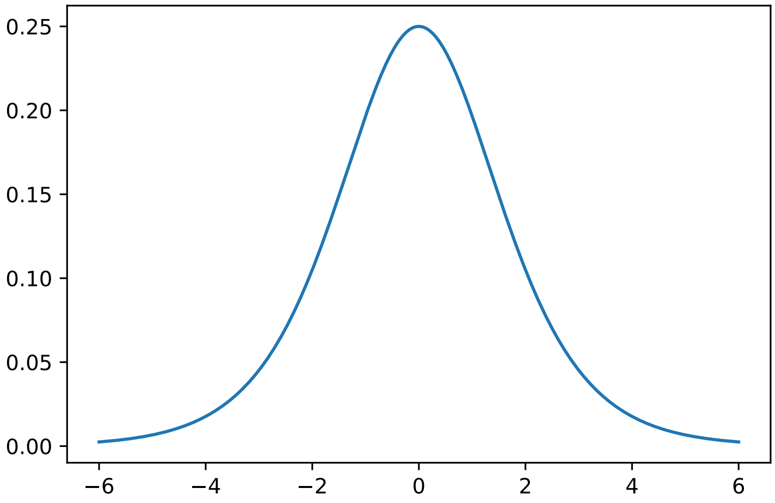


Figure 2.8. Figure of derivative of sigmoid function

$$S'(x) = S(x)(1 - S(x))$$

.

The derivative of the activation function can cause the problem of vanishing gradients. Let us demonstrate what can occur when the sigmoid is used as an activation function.

We can observe that the maximum value for the derivative function is 0.25, which is a low number. It is important to note that the number produced by the derivative of the activation function is then used in the chain rule in the backpropagation, where the goal is to minimise loss. Multiplying several low numbers leads to a number close to zero, which is called the vanishing gradient problem. A vanishing gradient causes the weights not to be updated, which means that the network will stop learning. We can prevent the vanishing gradient problem by using different activation functions, but we must be careful because other activation functions, such as ReLU, can result in an exploding gradient problem, which is the opposite of the vanishing gradient problem. Note that LSTM uses a hyperbolic tangent for cell activation, and the sigmoid activation function is usually used to activate the output of the node. It is important to note that, thanks to the LSTM architecture, the problem of vanishing gradient does not occur even though the sigmoid function is used as an activation function.

2.2.2 Extensions

So far, we have discussed the “basic” seq2seq model. Several extensions to the seq2seq model increase its accuracy and robustness. Let us discuss attention and transformer.

The problem with the seq2seq model is that the encoder outputs a fixed-size vector. That is, the entire input sentence (sequence of words) is compressed into a fixed-sized vector as the input for the decoder. We can call this a bottleneck because we give limited information about the input sequence. The decoder may need to “focus” on different parts of the encoder in each step, and using a fixed size vector will not allow this. Vanishing information can cause performance problems for long sentences that are squeezed into a single fixed-size vector. The attention focuses on this issue and proposes that the decoder have access to every state of the encoder.

Let us mention two attention mechanisms proposed by Bahdanau [30] and Luong [31]. These mechanisms are one of the most well-known and used attention mechanisms. Bahdanau and Luong models differ in the encoder used, the attention score, and when the attention is applied. Bahdanau model uses a bidirectional encoder, multilayer perceptron as an attention score, and the attention is applied between decoder steps. On the other hand, the Luong model uses a “simple” encoder, a bilinear function as an attention score, and the attention is applied after the RNN decoder step before prediction.

Attention mentioned above deals with “communication” between the encoder and the encoder, but if we extend the encoder and the decoder with attention (called self-attention), we get a transformer. The transformer can be summarised as “Attention for everyone”. With attention, we extend the encoder, decoder, and communication between the encoder and decoder. This approach led to several successful language models, such as BERT (Bidirectional Encoder Representations from Transformers) [32] by Google. The BERT is an English language model from 2018, trained for two tasks - to place missing words in the sentence and predict the following sentence. The Czech equivalent for BERT is Small-E-Czech[6] by Seznam, which contributed to increasing the quality of search results and can handle typographical corrections. Compared to BERT, Small-E-Czech has fewer weights, about 14 million. BERT, on the other hand, has 110 million weights. The number of weights might be disputable because larger networks

[6] https://github.com/seznam/small-e-czech

usually achieve higher accuracy, but their size is larger, and training and evaluation are computationally intensive.

Chapter 3
Implementation

The proposed implementation in this thesis comes from a trend in mathematical word problem solvers, which is described in the previous subchapter. Let us take a closer look at the implementation of the solver.

For the implementation, the Python programming language is used. The motivation for this programming language is that Python is standard for the latest solvers due to several machine learning libraries such as PyTorch and TensorFlow.

3.1 Preprocessing

The first thing that has to be taken care of is the solver's input and output data. The essential raw data from the dataset are word problems, equations, and results. We will consider a template for word problems and equations if we can not directly map numbers from a word problem to an equation. Template usage will be considered only during the training part. In the test part, the solver has to be capable of handling this situation on his own, which will be discussed further in the thesis.

An example of an input to the solver in the training part.

Word problem:

```
U školy roste 22 břízek, 8 lip a 15 smrčků. Kolik stromů roste u školy?
```

Word problem translation into English:

```
The school has 22 birches, 8 lindens, and 15 spruces.
How many trees grow at school?
```

Equation:

```
22 + 8 + 15
```

Result:

```
45
```

Additional information in case direct mapping is not possible.

Word problem without deterministic direct mapping:

```
Babička sbírala na zahrádce první jahody. Sebrala jich 30.
Radkovi dala 8 a Jitce také 8 jahod. Kolik jahod zůstalo babičce?
```

Equation without direct deterministic mapping:

```
30 - 8 - 8
```

Word problem template for word problem:

```
Babička sbírala na zahrádce první jahody. Sebrala jich NUM1.
Radkovi dala NUM2 a Jitce také NUM3 jahod. Kolik jahod zůstalo babičce?
```

Equation template for word problem:

```
NUM1 - NUM2 - NUM3
```

As can be seen, the word problem template and the equation template have substituted numbers with constants NUMY, where Y is the order number. Note that numbers that are not used in an equation can be substituted, for example, by NUMX, where X is not an order number. The substitution is essential for solvers because the equation solving the word problem will remain the same no matter what specific numbers are in the word problem. We can easily substitute numbers with constants because, in our dataset, we have two categories of word problems. The one where we can make direct mapping between word problems and equation numbers and after that assign every mapping a constant. The second category of word problems does not have direct mapping, but it has fields *word_problem_CZ_template*, *equation_CZ_template* and *mapping* containing information about the substitution. The question of how to make a substitution during the evaluation part will be discussed in the following chapters.

The generalisation above has strong prospects, but we can generalise even more. The motivation for the generalisation is to shrink the input and the output sequence space to cover more word problems and make the solver as general as possible. We do not have to stop only at the substitution of numbers. Some word classes can be substituted to cover general word problem cases, but other word classes should not be substituted as they can have a massive impact on solving a word problem. Another possible substitution is to replace words by their lemmas. This type of substitution should be less harmless than word class substitution, which can substitute essential words with a constant, resulting in loss of meaning. Syntactic analysis of the word problem is needed to perform word classes substituting or substituting words by their lemmas. One of the best tools for syntactic analysis is UDPipe [33]. It is a tool that can perform tagging, lemmatization, and syntactic analysis, which can all be used for any substitution.

The UDPipe provides the REST service[1] and the Python library[2] to perform syntactic analysis. Several inputs can be passed to UDPipe. We will consider the most intuitive one, plain text. The output, on the other hand, is in CoNLL-U format[3] containing all information about syntactically analyzed input.

Although UDPipe is one of the best syntactic analyzers for the Czech language (supporting other languages), it is not flawless. That is why it is not a good idea to rely only on syntactic analyzers. It should always be taken into account that they can produce errors. Fortunately for our approach, we will rely on the UDPipe results minimally.

[1] https://lindat.mff.cuni.cz/services/udpipe/

[2] https://pypi.org/project/ufal.udpipe/

[3] https://universaldependencies.org/format.html

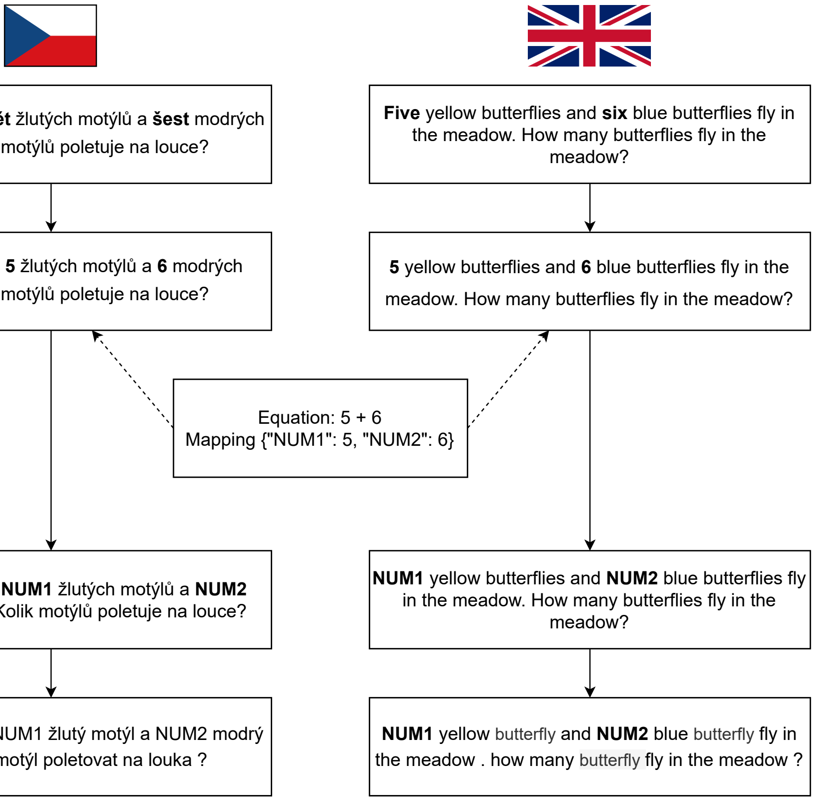
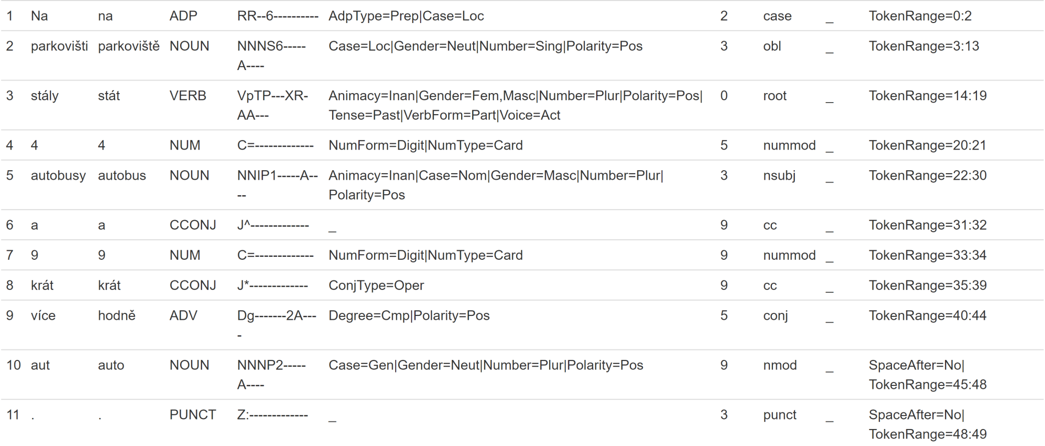

Na louce poletuje **pět** žlutých motýlů a **šest** modrých motýlů. Kolik motýlů poletuje na louce?
Five yellow butterflies and **six** blue butterflies fly in the meadow. How many butterflies fly in the meadow?
Na louce poletuje **5** žlutých motýlů a **6** modrých motýlů. Kolik motýlů poletuje na louce?
5 yellow butterflies and **6** blue butterflies fly in the meadow. How many butterflies fly in the meadow?
Equation: 5 + 6
Mapping {"NUM1": 5, "NUM2": 6}
Na louce poletuje **NUM1** žlutých motýlů a **NUM2** modrých motýlů. Kolik motýlů poletuje na louce?
NUM1 yellow butterflies and **NUM2** blue butterflies fly in the meadow. How many butterflies fly in the meadow?
na louka poletovat NUM1 žlutý motýl a NUM2 modrý motýl . kolik motýl poletovat na louka ?
NUM1 yellow butterfly and **NUM2** blue butterfly fly in the meadow . how many butterfly fly in the meadow ?

Figure 3.1. The figure shows preprocessing the input word problem. Numbers are substituted by variables, words are substituted by their lemmas.

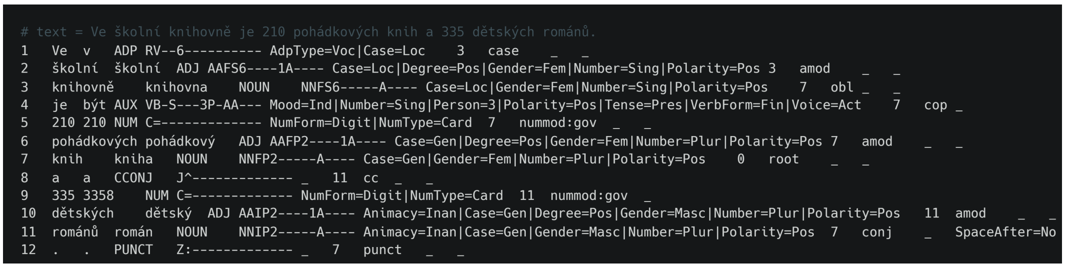


1	Na	na	ADP	RR--6----------	AdpType=Prep|Case=Loc	2	case	_	TokenRange=0:2
2	parkovišti	parkoviště	NOUN	NNNS6-----A----	Case=Loc|Gender=Neut|Number=Sing|Polarity=Pos	3	obl	_	TokenRange=3:13
3	stály	stát	VERB	VpTP---XR-AA---	Animacy=Inan|Gender=Fem,Masc|Number=Plur|Polarity=Pos|Tense=Past|VerbForm=Part|Voice=Act	0	root	_	TokenRange=14:19
4	4	4	NUM	C=-------------	NumForm=Digit|NumType=Card	5	nummod	_	TokenRange=20:21
5	autobusy	autobus	NOUN	NNIP1-----A----	Animacy=Inan|Case=Nom|Gender=Masc|Number=Plur|Polarity=Pos	3	nsubj	_	TokenRange=22:30
6	a	a	CCONJ	J^-------------	_	9	cc	_	TokenRange=31:32
7	9	9	NUM	C=-------------	NumForm=Digit|NumType=Card	9	nummod	_	TokenRange=33:34
8	krát	krát	CCONJ	J*-------------	ConjType=Oper	9	cc	_	TokenRange=35:39
9	více	hodně	ADV	Dg-------2A----	Degree=Cmp|Polarity=Pos	5	conj	_	TokenRange=40:44
10	aut	auto	NOUN	NNNP2-----A----	Case=Gen|Gender=Neut|Number=Plur|Polarity=Pos	9	nmod	_	SpaceAfter=No| TokenRange=45:48
11	.	.	PUNCT	Z:-------------	_	3	punct	_	SpaceAfter=No| TokenRange=48:49

Figure 3.2. The figure shows an example of syntactic analysis for the sentence "Na parkovišti stály 4 autobusy a 9 krát více aut." outputted by UDPipe REST service.

We will focus only on lemmas and parts of speech, meaning we do not expect UDPipe errors to cause many issues for us.

Using a syntactic analyzer provides another benefit. In the word problem examples, there are numbers written using digits, but the input to the word problem solver can contain a number written by words, which is valid from the grammar point of view. Thanks to syntax analysis, the lemma of the word number can be obtained and, using a defined dictionary, it can be converted to a number written by digits.



```
# text = Ve školní knihovně je 210 pohádkových knih a 335 dětských románů.
1   Ve  v   ADP RV--6---------- AdpType=Voc|Case=Loc    3   case    _   _
2   školní  školní  ADJ AAFS6----1A---- Case=Loc|Degree=Pos|Gender=Fem|Number=Sing|Polarity=Pos 3   amod    _   _
3   knihovně    knihovna    NOUN    NNFS6-----A---- Case=Loc|Gender=Fem|Number=Sing|Polarity=Pos    7   obl _   _
4   je  být AUX VB-S---3P-AA--- Mood=Ind|Number=Sing|Person=3|Polarity=Pos|Tense=Pres|VerbForm=Fin|Voice=Act    7   cop _
5   210 210 NUM C=------------- NumForm=Digit|NumType=Card  7   nummod:gov  _   _
6   pohádkových pohádkový   ADJ AAFP2----1A---- Case=Gen|Degree=Pos|Gender=Fem|Number=Plur|Polarity=Pos 7   amod    _   _
7   knih    kniha   NOUN    NNFP2-----A---- Case=Gen|Gender=Fem|Number=Plur|Polarity=Pos    0   root    _   _
8   a   a   CCONJ   J^------------- _   11  cc  _   _
9   335 3358    NUM C=------------- NumForm=Digit|NumType=Card  11  nummod:gov  _
10  dětských    dětský  ADJ AAIP2----1A---- Animacy=Inan|Case=Gen|Degree=Pos|Gender=Masc|Number=Plur|Polarity=Pos   11  amod    _   _
11  románů  román   NOUN    NNIP2-----A---- Animacy=Inan|Case=Gen|Gender=Masc|Number=Plur|Polarity=Pos  7   conj    _   SpaceAfter=No
12  .   .   PUNCT   Z:------------- _   7   punct   _   _
```

Figure 3.3. The figure shows an error that UDPipe made. The input sentence is “Ve školní knihovně je 210 pohádkových knih a 335 dětských románů”. As we can observe the UDPipe changes lemma for number 335 to 3358. This error occurs for UDPipe model *czech-pdt-ud-2.5-191206.udpipe* (the latest available model to download).

3.2 Training model

The previous subchapter summarizes the input and output data preprocessing for a solver. This subchapter will be dedicated to the solver training. As mentioned earlier, our approach is based on the seq2seq model and modifications derived from the latest solvers.

The seq2seq model was introduced in detail in Subchapter 2.2, and as we found, the model is complex. It consists of an encoder and decoder, which can have different architectures using GRU or LSTM. Other extensions can be added, such as the attention mechanism, greedy or beam search in the decoder, and parameters for RNN (hidden size, drop out, teacher force ratio, and others). The PyTorch framework presents a tutorial[4] dedicated to translating French into English. Let us have the model from the tutorial as a foundation for the word problem solver presented in this thesis and adapt and extend it to solve word problems.

The PyTorch tutorial presents a simple encoder based on GRU, reproducing a single output vector with hidden values. The architecture of the encoder can be seen in Figure 3.4.

The tutorial proposes two decoders, a simple one and a decoder with an attention mechanism. Both decoders use GRU as an RNN architecture. The visualization of the simple decoder can be seen in Figure 3.5. As can be observed, the decoder does not hide anything complex. On the contrary, the decoder using an attention mechanism becomes more complex, as shown in Figure 3.6.

We can make several observations for the model proposed in the tutorial. One observation is that the model is trained using stochastic gradient descent, which is worth considering to try out the different approaches to training, batch, and mini-batch. Both decoders and encoders use GRU as part of their architecture. LSTM, Bi-LSTM are worth trying out. Lastly, the model deals with translation and can also be used to solve word problems without any modifications, but we can easily make some improvements. The seq2seq model uses so-called “greedy decoding”, which means that it picks the token with the highest probability. Unfortunately, this may not always lead to picking semantically/syntactically valid tokens, resulting in nonsense equations.

For example, the following sequence can be obtained:

```
NUM1 NUM2 / + NUM3
```

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

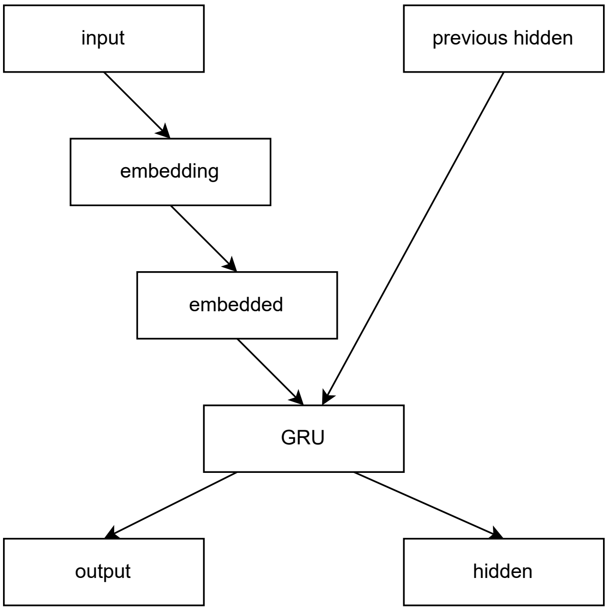


Figure 3.4. The encoder proposed in PyTorch tutorial.

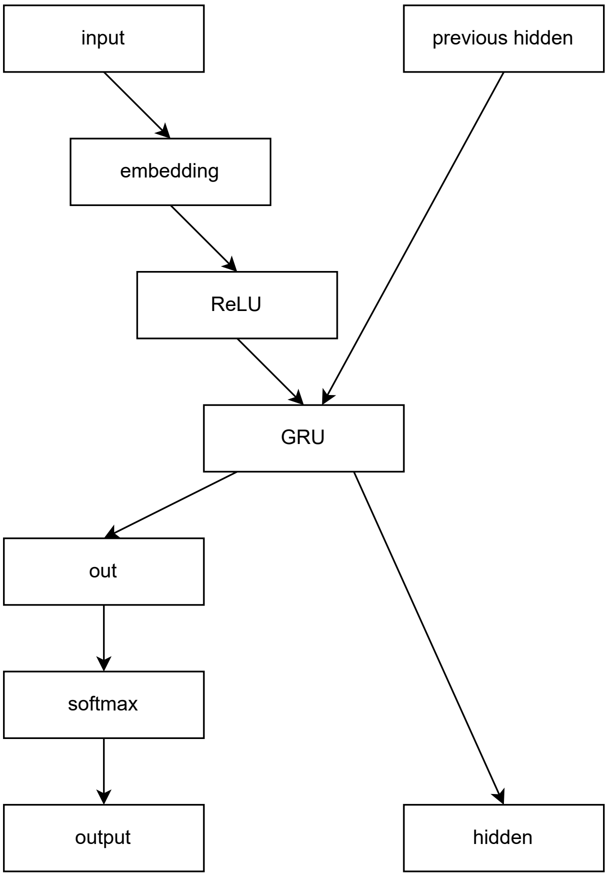


Figure 3.5. The simple decoder proposed in PyTorch tutorial.

DNS proposes a solution in the form of a set of rules applied before “greedy decoding”, before the activation function is called, which results in the zero probability of placing the token incorrectly.

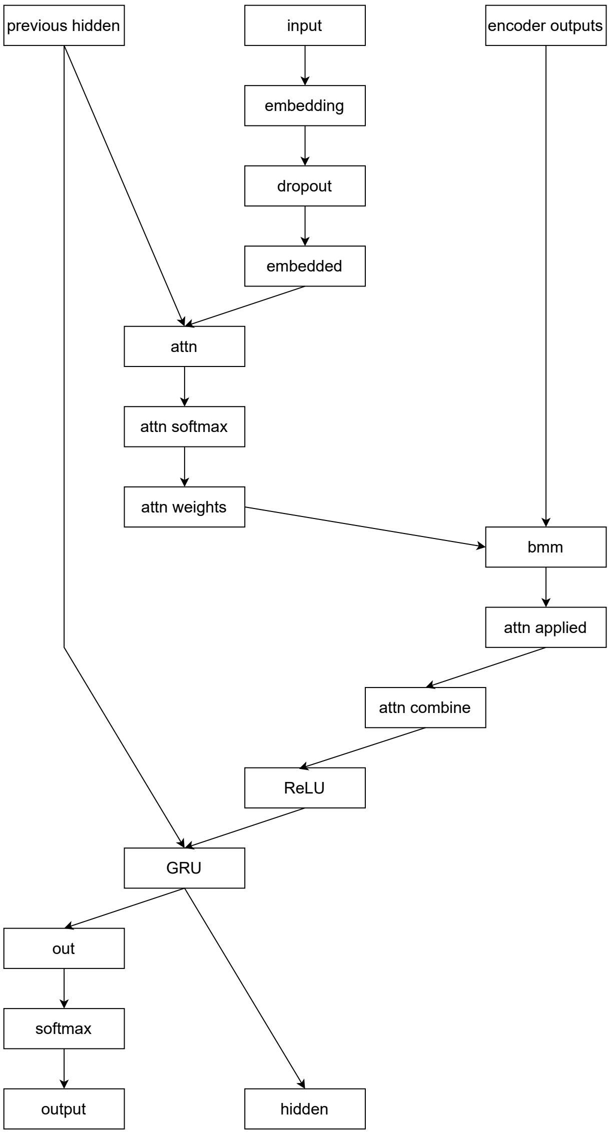


Figure 3.6. The decoder with attention mechanism proposed in PyTorch tutorial.

The adjusted rules are the following (r_{t-1} is a token in the previous position):

- If r_{t-1} is in $\{ +, -, *, / \}$, then r_t will not be in $\{ +, -, *, /,) \}$
- If r_{t-1} is a number, then r_t will not be a number
- If r_{t-1} is (, then r_t will not be in $\{ +, -, *, /,), (\}$
- If r_{t-1} is), then r_t will not be a number and not in $\{), (\}$

Binary vector ρ_t is created using these rules and then used as a mask to filter out the unwanted candidates for a token before the activation function. The following equation shows how a binary vector ρ_t (mask) is applied to the softmax function that produces the probability distribution, where the invalid tokens for position t have zero probability.

$$P(r_t|h_t) = \frac{\rho_t \odot e^{h_t^T W^s}}{\sum \rho_t \odot e^{h_t^T W^s}}$$

One of the observations made in this thesis is that during training, the length of the sequence is known, but this does not apply to the testing part. The assumption that can be made is that the length of the valid sequence will always be an odd number not less than 3. The reason for this is that, in this case, math operations are always binary operations, which means that the operation requires two input values. Then there are parentheses, which always have even numbers starting with 0. The length of the equation is then the sum of input values (numbers), math operations, and parentheses, which is $(2k{+}1){+}2l$, where $2k{+}1; k \in \mathbb{N}^{+}$ is the number of math operations and numbers and $2l; l \in \mathbb{N}$ is the number of parentheses. The resulting sum is then $(2k+1)+2l = 2(k+l)+1 = 2m+1;\ m \in \mathbb{N}^{+}$. The assumption above is valid only when we know that all math operations are binary operations. The assumption does not hold for the following equation because ’−’ (unary minus) is not a binary operation.

$$-NUM1 + NUM2$$

We can replace “greedy decoding”, which picks the most probable token, with “beam search”, which tracks several of the most probable hypothesis, and then selects the most probable one, and we can further extend it with requirements such as that the equation is valid, or application of rules mentioned before.

3.3 Evaluation

When we train a model, we are ready to evaluate the test dataset, but a challenge must be solved before that. The challenge is to map the number of word problems. Since we were unable to develop a deterministic mapping during training, we used an attribute that holds the information. Unfortunately, this cannot be done in the evaluation part because we do not want to provide a model with this information. It would be too trivial for the model to evaluate the word problem and it would not be able to solve some word problems without additional information. The DNS deals with this issue using the SNI (Significant number identification) model. The SNI is a binary classification based on LSTM that determines if a number is significant in a chunk of a word problem. The number is considered to be “significant” if it is used in the equation that solves the assignment.

```
Word problem in Czech:
Tobiáš dostal za úkol vypočítat 2 cvičení z matematiky. V prvním byly 4
sloupečky po 5 příkladech a ve druhém 3 sloupečky po 4 příkladech.
Kolik musel Tobiáš vypočítat příkladů?

Word problem in English:
Tobias was given the task of calculating 2 exercises in mathematics.
There were 4 columns of 5 examples in the first and the second 3 columns
of 4 examples. How many examples did Tobias have to calculate?

Equation:
4 * 5 + 3 * 4
```

In the word problem above, we have five numbers, and, as we can see, the equation solving this word problem needs four of them (number 2 is not a part of the equation).

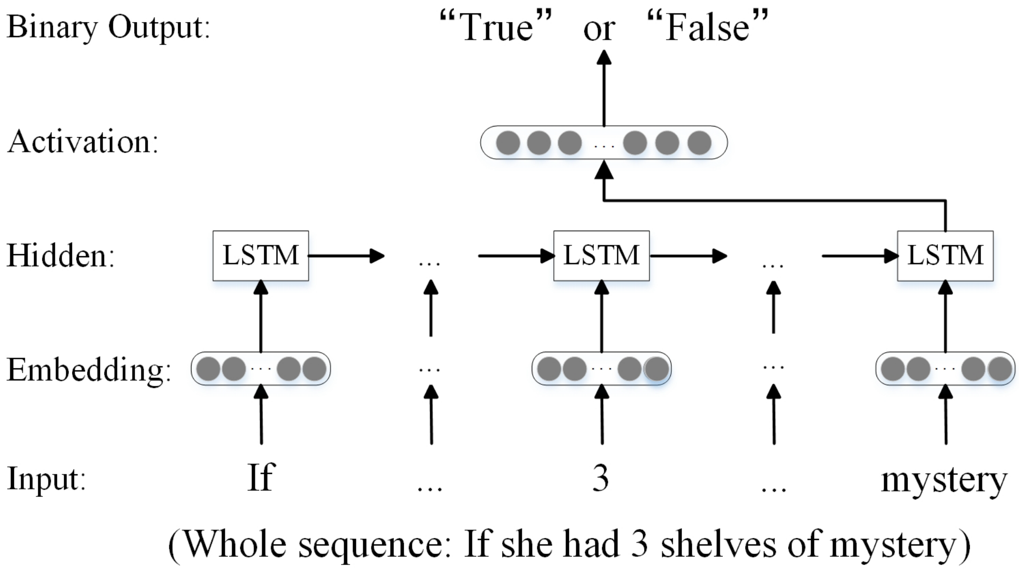


Figure 3.7. Visualization of SNI (Significant number identification) from [7].

SNI from DNS does not seem optimal because it only evaluates a part of the problem. What if the information, when the number is significant, is hidden in a different chunk? That is why a modified SNI is required. The modification of SNI proposed in this thesis works on the whole word problem with all numbers substituted by constants. The output sequence consists of numbers, essential constants. We can apply a similar trick to that proposed by DNS. When the decoder decodes the output sequence, we can force it to return unique tokens. Let us say that the decoder's first token is *NUM1*, which means that the token *NUM1* is significant, and by this, we can deduce that the following tokens will not be *NUM1* because we already know that the token *NUM1* is significant.

Although we will be able to find "significant" numbers for word problems and pass replaced numbers by constants to the model, we will have to determine how to evaluate the model's output. We cannot determine precisely if the equation output by the model is correct without taking into account the mapping found by SNI because it could happen that the ground truth equation would be the same as the predicted equation, but the constants in the equation would point to different numbers. One of the solutions is to replace constants with numbers from mapping and evaluating the equation. However, this approach is suboptimal because the results can be the same, but the mapping can be different. Therefore, the equation and the mapping should be checked during the evaluation test dataset.

Chapter 4
Experiments

This chapter describes two experiments. The first experiment is focused on different preprocessing templates and their impact on accuracy. Afterward, we use the best performing preprocessing template from the first experiment in the second experiment where we work with various seq2seq architectures and compare their accuracy with the results from the first experiment.

The following libraries and tools were used in the implementation:

- Python 3.9.5
- PyTorch 1.10.0
- CUDA 11.3.1
- UDPipe 2 (czech-pdt-ud-2.5-191206)

All models for the experiment were trained on GPU servers of the Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague. The specification of the GPU used for training is the following:

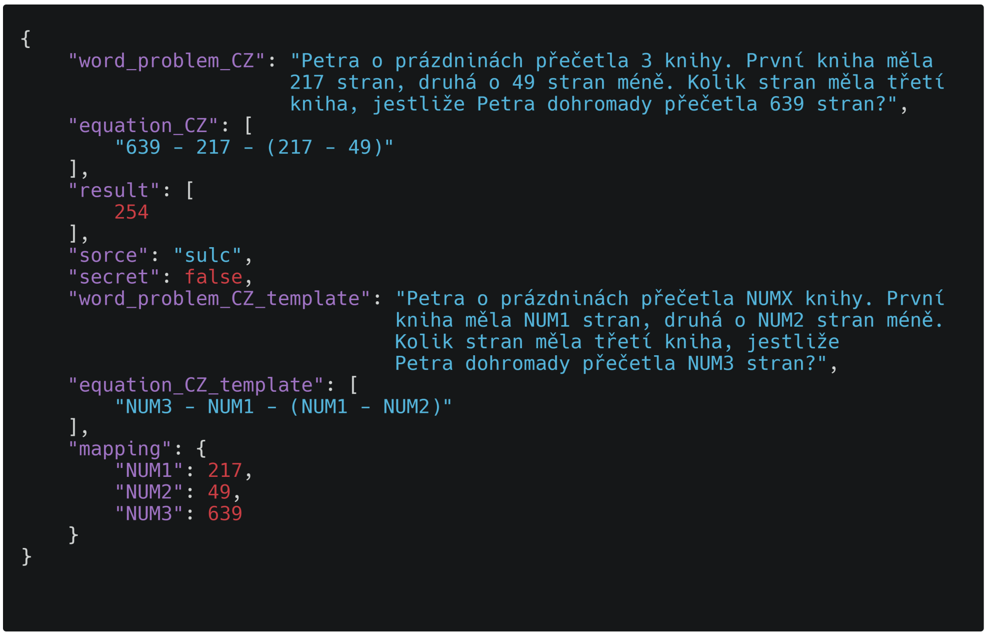
- NVIDIA GTX 1080Ti
- Memory: 11178MiB

4.1 Preprocessing templates experiments

Let us present four preprocessing templates, but before that, we have to clarify that these templates are something different from the *word_problem_CZ_template*, which helps us in training to find the correct mapping of numbers. These preprocessing templates were already mentioned in Subchapter 3.1. Their purpose is to generalise the training record for the word problem, resulting in higher model accuracy.

The four preprocessing templates are:

- NoSub
- Simple
- Smarter
- Ultimate



```
{
    "word_problem_CZ": "Petra o prázdninách přečetla 3 knihy. První kniha měla
                        217 stran, druhá o 49 stran méně. Kolik stran měla třetí
                        kniha, jestliže Petra dohromady přečetla 639 stran?",
    "equation_CZ": [
        "639 - 217 - (217 - 49)"
    ],
    "result": [
        254
    ],
    "sorce": "sulc",
    "secret": false,
    "word_problem_CZ_template": "Petra o prázdninách přečetla NUMX knihy. První
                                 kniha měla NUM1 stran, druhá o NUM2 stran méně.
                                 Kolik stran měla třetí kniha, jestliže
                                 Petra dohromady přečetla NUM3 stran?",
    "equation_CZ_template": [
        "NUM3 - NUM1 - (NUM1 - NUM2)"
    ],
    "mapping": {
        "NUM1": 217,
        "NUM2": 49,
        "NUM3": 639
    }
}
```

Figure 4.1. The word problem "Petra read 3 books during the holidays. The first book had 217 pages, the second 49 pages less. How many pages did the third book have if Petra read 639 pages together?" demonstrates how a template is used.

We will consider word problem and equation from Figure 4.1 as an input to the four preprocessing templates and show how the preprocessing affects the word problem and describe what the template does.

NoSub template does nothing at all. This template takes the word problem and equation and returns them unchanged. This template demonstrates how preprocessing is crucial and how even tiny preprocessing can boost accuracy. Although we consider this template to be the one that will not work well in the test set, it still has an enormous advantage over other templates. It does not need to detect significant numbers because it does not substitute them. Not being dependent on significant number identification might be an advantage because the solver depends only on itself, so it avoids the situation when the significant number identification wrongly identifies a significant number, resulting in the solver's wrong solution. Another positive side effect is that it should be faster than templates using significant number identification.

```
Word problem:
Petra o prázdninách přečetla 3 knihy . První kniha měla 217 stran , druhá
o 49 stran méně . Kolik stran měla třetí kniha , jestliže Petra dohromady
přečetla 639 stran ?

Equation:
639 - 217 - ( 217 - 49 )

Mapping:
{}
```

Simple template provides one of the essential preprocessing procedures. It substitutes numbers with constants, resulting in basic generalisation when we do not mind about numbers presented in the word problem because they do not affect the equation which solves it. As mentioned above, this template must identify significant numbers in the test set to use the trained significant number identification model. We substitute numbers that are not significant with constant string *NUMX* , as can be seen in the example below.

```
Word problem:
Petra o prázdninách přečetla NUMX knihy . První kniha měla NUM1 stran ,
druhá o NUM2 stran méně . Kolik stran měla třetí kniha , jestliže
Petra dohromady přečetla NUM3 stran ?

Equation:
NUM3 - NUM1 - (NUM1 - NUM2)

Mapping:
{'NUM1': 217, 'NUM2': 49, 'NUM3': 639}
```

The Czech language is a so-called fusional language, which means that it does inflection, i.e. word formation. Unfortunately, inflection makes solving word problems more challenging because it does not bring additional information that would help solve the word problem. Sometimes similar words mean the same thing. Smarter template should handle this issue. As Simple template, it substitutes numbers with constants, but it also substitutes words with their lemmas, which should solve the challenge of inflection.

```
Word problem:
Petra o prázdniny přečíst NUMX kniha . první kniha mít NUM1 strana ,
druhý o NUM2 strana málo . kolik strana mít třetí kniha , jestliže
Petra dohromady přečíst NUM3 strana ?

Equation:
NUM3 - NUM1 - (NUM1 - NUM2)

Mapping:
{'NUM1': 217, 'NUM2': 49, 'NUM3': 639}
```

Unlike the previous templates, the last template uses syntactic analysis obtained using the UDPipe program. It uses information about the part of speech of each word. The idea is that not all parts of speech are essential and that we can substitute some parts of speech with constants, such as numbers. So, the Ultimate template is an extension of the Smarter template with the difference that we replace some parts of speech with constants. For testing purpose, we decided to replace nouns and proper nouns by NOUN and PROPN, respectively. These are not only parts of speech that can be substituted, but in our opinion, it is possible to substitute some other less important parts as well, such as ADJ (adjective), but it is essential to mention that this substitution comes with a penalty which is losing information. Some parts of speech can be crucial to solving a word problem, such as VERB (verb) or ADV (adverb). On the other hand, we gain generalisation (simplification) of word problems.

```
Word problem:
PROPN1 o NOUN1 přečíst NUMX NOUN2 . první NOUN2 mít NUM1 NOUN3 , druhý o
NUM2 NOUN3 málo . kolik NOUN3 mít třetí NOUN2 , jestliže PROPN1
```

```
dohromady přečíst NUM3 NOUN3 ?

Equation:
NUM3 - NUM1 - (NUM1 - NUM2)

Mapping:
{'NUM1': 217, 'NUM2': 49, 'NUM3': 639}
```

The templates mentioned above should show us how a generalisation of the word problem can help us find a robust solver. We expect that some templates may overfit, and these solvers should be those that do not generalise enough, such as NoSub template.

For this experiment, we provided all templates with the same conditions. We trained the four templates in 20 epochs on the same training data consisting of 18,834 word problems, which were randomly chosen from the created dataset. The architecture proposed in the PyTorch tutorial, which we mentioned before, was used for training. Unfortunately, the architecture did not allow us to use the mask that uses DNS. The reason is that the architecture uses negative logarithmic likelihood, logarithmic softmax, and mask force values to have zero probability. The logarithm of zero is minus infinity, resulting in a loss that will be plus infinity, making the model unable to be well trained. We evaluated models using the four proposed templates on the same test dataset of size 4,709 samples. Word problems from the *matikain* dataset were excluded from the train and test dataset to show how the solver performs on a small dataset and which also includes types of word problems that were not present in the training set. We consider one more dataset for comparison - the *reischigova* dataset from bachelor thesis [11], which contains 499 word problems (396 of them are part of training data and 103 of them are part of testing data).

Some templates need significant number identification for evaluation to replace numbers with constants. We trained a significant number identification model in 20 epochs on the different training data we used for the training templates. The train dataset contains 18,854 word problems, and the test dataset contains 4,714 word problems. We expect Ultimate template to be the best performing template, and that is why we use it in significant number identification in the preprocessing part.

As we can observe in Figure 4.2, the advanced templates, preprocessing the word problem, achieved a lower loss than the naive NoSub template. However, the loss started to increase. An unsuitable learning rate can cause an increase in training loss, which is probably our example. We used the 0.01 learning rate for the experiment, so using the next iteration's lower learning rate for the templates would be wise. Interestingly, the learning rate was improper only for models that used templates that modified the input sequence.

The experiment revealed that, as expected, templates significantly impact the model's accuracy. An interesting observation is that, on the testing dataset, Smarter template performed better than Ultimate template, which is unexpected. However, we will consider Ultimate to be the best performing template because it is more accurate when considering the absolute number of solved word problems.

From the results and examination of the wrong-solved word problems, we identify two major error categories: syntactically incorrect equations and wrong equations. The number of syntactically incorrect equations could be reduced using the trick presented

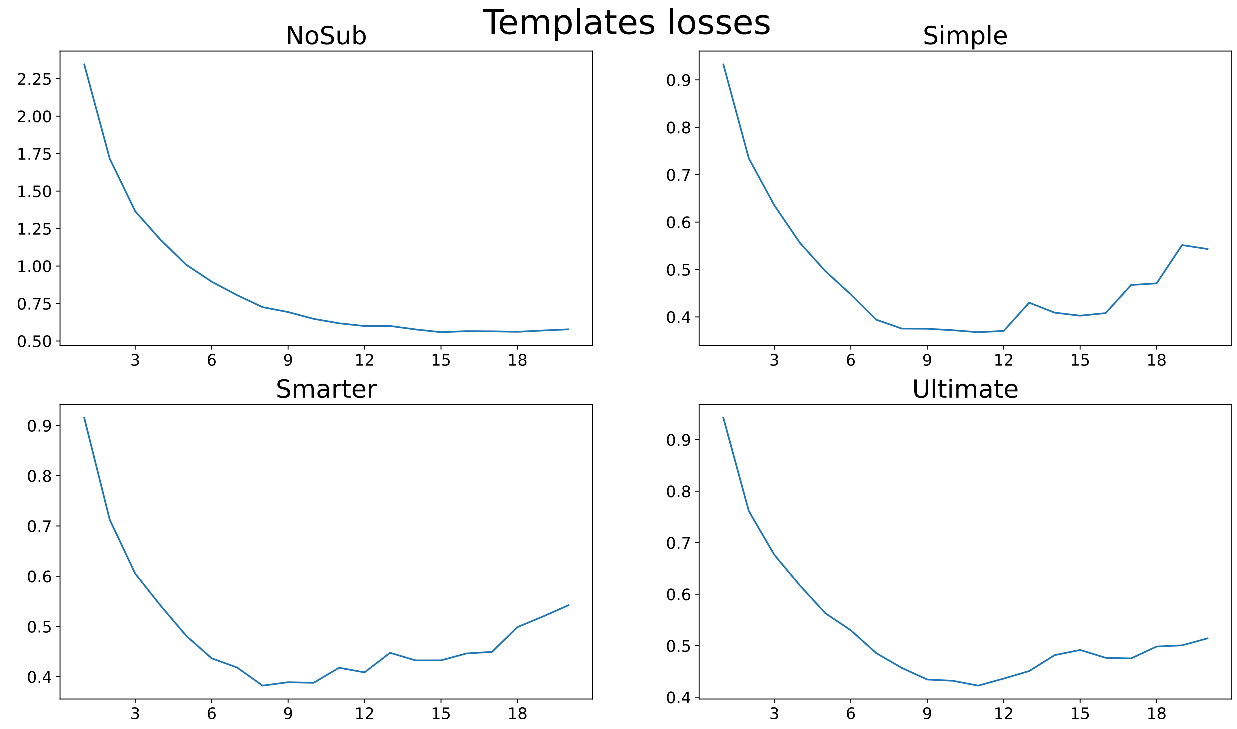


Figure 4.2. The figure of template losses. X-axis are epochs, Y-axis are losses.

dataset	NoSub	Simple	Smarter	Ultimate
training	41.6%/45.67%	42.68%/46.41%	47.16%/51.99%	**48.64%**/**52.60%**
testing	14.7%/21.21%	33.45%/38.97%	**37.06%**/**43.39**%	35.17%/42.11%
reischigova	48.1%/53.9%	51.9%/53.31%	58.12%/59.12%	**68.14%**/**68.93%**
matikain	12.00%/**16.00%**	**16.00%**/**16.00%**	**16.00%**/12.00%	**16.00%**/**16.00%**

Table 4.1. Template models accuracy on the datasets with ground truth information about significant numbers on the datasets. As we mentioned in Subchapter 3.3, there are two options to compute the accuracy — comparing equations or comparing results. In the table, the first number is the accuracy of comparing equations, and the second is the accuracy of comparing results.

dataset	NoSub	Simple	Smarter	Ultimate
training	41.6%/45.67%	41.32%/40.28%	45.63%/45.75%	**47.05%**/**46.02%**
testing	14.7%/21.21%	32.63%/34.27%	**36.22%**/**38.52%**	34.06%/37.07%
reischigova	48.1%/53.9%	51.50%/50.30%	57.91%/56.11%	**67.73%**/**64.72%**
matikain	12.00%/**16.00%**	**16.00%**/**16.00%**	**16.00%**/**16.00%**	**16.00%**/**16.00%**

Table 4.2. Template models accuracy on the datasets when using SNI. The accuracy for NoSub template did not change, because it does not need to use SNI model. The first number is the accuracy of comparing equations, and the second is the accuracy of comparing results.

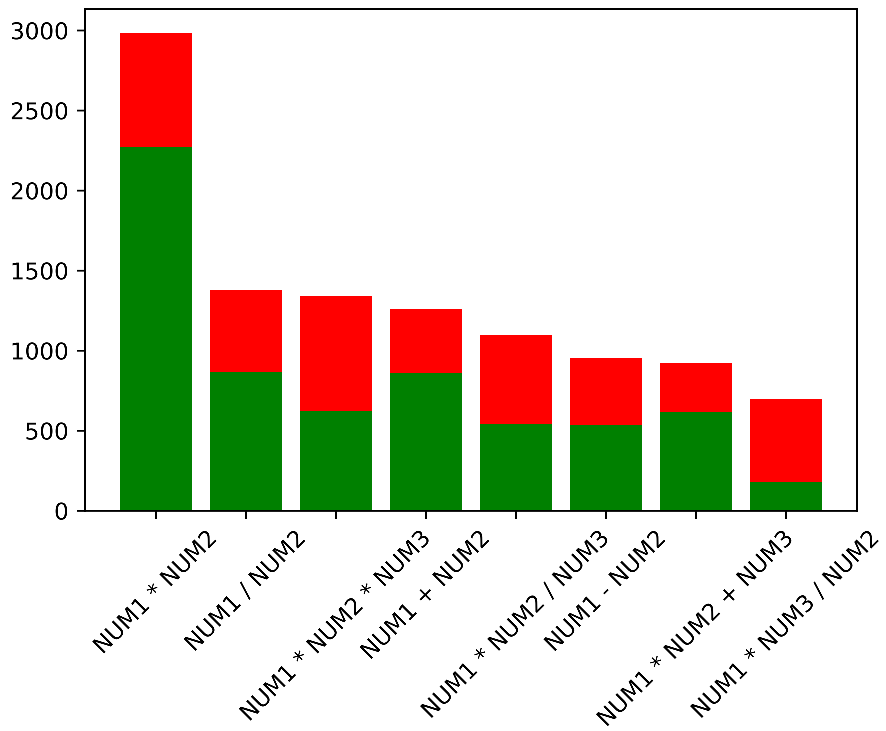


Figure 4.3. The figure shows accuracy of the eight most common equations from the training and testing dataset. The green area shows the number of correctly solved word problems, and the red area shows the number of wrongly solved word problems. The results are shown for Ultimate, using equation comparison with SNI. The overall accuracy of the eight most common equations is 61.16%.

in DNS, there would not have to be any syntactically incorrect word problems in the best scenario. On the other hand, minimising the wrong equations is a challenge and could be done using a more robust architecture or more advanced preprocessing. The distributions of two error categories in the dataset can be seen in Figure 4.3.

An example of a syntactically incorrect equation on the output:

```
Word problem:
Jídelna nakoupila zpět 36 pytlů rýže a mouky.
Rýže je 60 kg na pytel a mouky 50 kg na pytel.
Počítejte, kolik kilogramů rýže a mouky jste nakoupili zpět?

Word problem in English:
The dining room bought back 36 bags of rice and flour.
Rice is 60 kg per bag, and flour is 50 kg per bag.
Count how many kilograms of rice and flour you bought back?

Ground truth equation:
NUM1 * (NUM2 + NUM3)

Equation from the model:
( * NUM1 * NUM1
```

An example of a wrong equation:

```
Word problem:
Poměr broskvoní a hrušní v sadu je 5:3.
Existuje 40 broskvoní a kolik hrušní?

Word problem in English:
The ratio of peaches and pears in the orchard is 5:3.
There are 40 peaches, and how many pears?

Ground truth equation:
NUM3 / (NUM1 / NUM2)

Equation from the model:
NUM3 * (NUM2 / NUM1)
```

From the above examples, we can observe that syntactically incorrect equations cannot be evaluated. The wrong equations can be evaluated, but they are not the same as the ground truth equations.

dataset	syntax error	incorrect equation	incorrect sum
training w/o SNI	2,759 (28.55%)	6,906 (71.45%)	9,665
testing w/o SNI	870 (28.52%)	2,180 (71.48%)	3,050
training + testing w/o SNI	3,629 (28.54%)	9,086 (71.46%)	12,715

Table 4.3. The table shows the error distribution for Ultimate template using the ground truth information about significant numbers.

template	model size [kB]
NoSub	30,994
Simple	26,689
Smarter	16,739
Ultimate	10,850

Table 4.4. The table shows the sizes of models using a specific template.

As we can observe in Figure 4.4, preprocessing using templates has, except for higher accuracy and generalisation, another advantage - the model size. We need to keep in mind that for Simple, Smarter, and Ultimate templates, we may also want a significant number identification model, which is of size 10,807kB, which means that the model of the largest size (together with significant number identification) is Simple template. However, Smarter and Ultimate templates are still smaller than NoSub template.

4.2 Training and evaluation on different architectures

Ultimate template achieved the best accuracy in the previous subchapter. That is why we used it in this experiment as a primary preprocessing template. The first experiment revealed that the learning rate was set too high and caused problems during model training. Therefore, this experiment will use a lower learning rate: 0.001. Another change in the architecture used in the second experiment is the usage of RNNs. The first experiment used a tutorial example of the seq2seq architecture, which used GRU as RNN for the encoder and decoder. We will use bidirectional LSTM with two layers in this experiment because our dataset is quite large. LSTM has more parameters than GRU, which we hope will result in higher accuracy. The PyTorch architecture uses well-known SGD, an iterative optimizer of objective function, as an encoder and decoder optimiser, and the architecture in the second experiment uses an Adam optimiser [34]. This change occurs because we expect the Adam optimiser to provide better learning, resulting in higher accuracy. Lastly, we mentioned that we could not use the trick with the mask presented by DNS for the previous experiment. Let us try to modify the trick with the mask. Instead of the vector ρ_t with ones and zeros, we propose to use ρ_t with ones and a penalisation, which is a real number in the interval $(0, 1)$. As we mentioned earlier, the architecture from the first experiment uses a negative logarithmic likelihood to compute the loss. The input of negative logarithmic likelihood is formed of the logarithmic probabilities of each class. If the probability is zero, the log-probability is minus infinity, resulting in an infinity loss if the ground truth class has zero probability. The idea behind this proposed modification is to penalise, making the probability of the class smaller, but not zero, to avoid an infinity loss. For this experiment, we use penalisation 0.5.

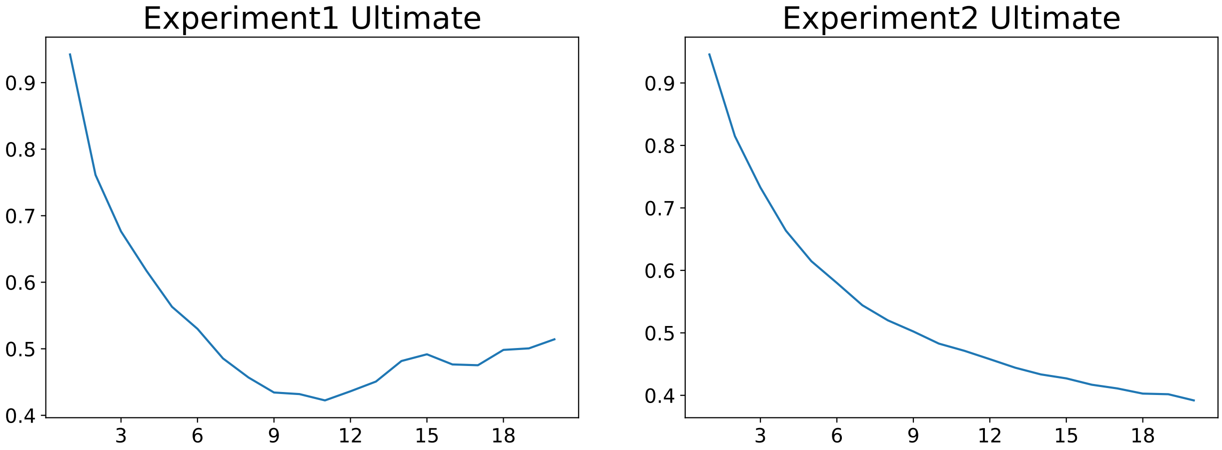


Figure 4.4. The figure of experiment losses. X-axis are epochs. Y-axis are losses.

From Figure 4.4 we can observe that our proposed architecture solved the problem of increasing the loss, which we faced in the first experiment.

dataset	Experiment1 Ultimate	Experiment2 Ultimate
training	48.64%/52.60%	**65.63%/74.71%**
testing	35.17%/42.11%	**47.63%/59.35%**
reischigova	68.14%/68.93%	**80.36%/81.36%**
matikain	**16.00%/16.00%**	12.00%/12.00%

Table 4.5. The model's accuracy with ground truth information about significant numbers on the datasets.

dataset	Experiment1 Ultimate	Experiment2 Ultimate
training	47.05%/46.02%	**62.40%/62.25%**
testing	34.06%/37.07%	**46.42%/50.96%**
reischigova	67.73%/64.72%	**77.95%/76.55%**
matikain	**16.00%/16.00%**	12.00%/12.00%

Table 4.6. The model's accuracy accuracy using SNI to identify significant numbers.

dataset	syntax error	incorrect equation	incorrect sum
training w/o SNI	149 (2.36%)	6,173 (97.64%)	6,322
testing w/o SNI	49 (2.01%)	2,387 (97.98%)	2,436
training + testing w/o SNI	198 (2.26%)	8,560 (97.74%)	8,758

Table 4.7. The error distribution for Ultimate template using ground truth information about significant numbers.

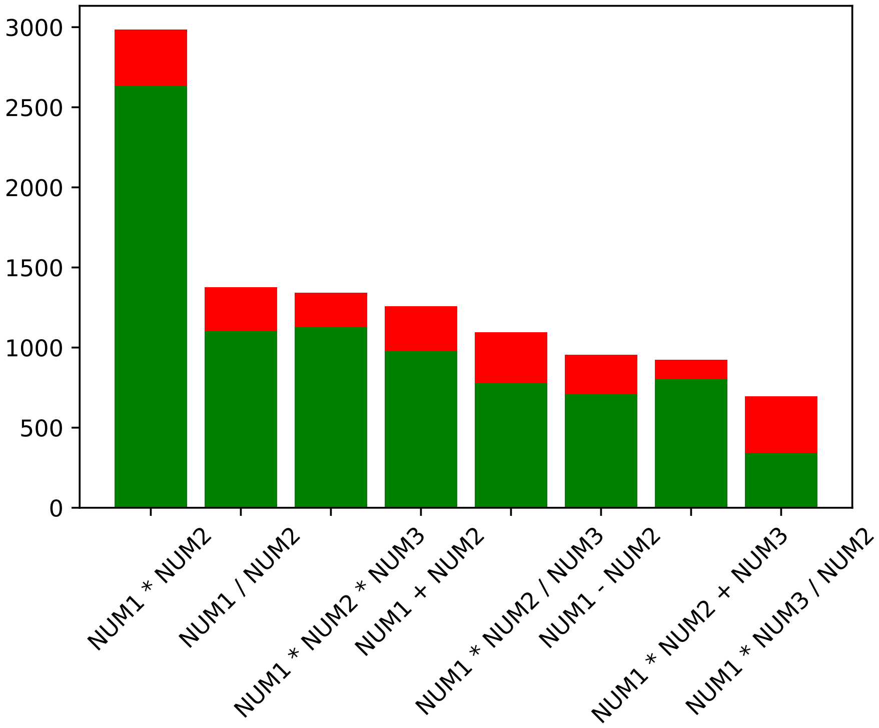


Figure 4.5. The figure shows the accuracy of the eight most common equations templates from the training and testing dataset. The green area shows the number of correctly solved word problems, and the red area shows the number of wrongly solved word problems. The experiment uses the model proposed in the second experiment, equation comparison with SNI. The overall accuracy of the eight most common equation templates is 79.90%.

Figures 4.5 and Figure 4.6 show that we were able to obtain higher accuracy in both scenarios (with ground truth information about significant numbers and with SNI). Unfortunately, the architecture proposed in this experiment performed worse for the *matikain* dataset, which need not to mean that it is worse for general word problems. There are 173 word problems in the testing set that are solved by an equation not present in the training set. The architecture of the second experiment was able to solve 14 of them. The architecture of the first experiment did not solve any such problem. After all, the proposed architecture has higher accuracy for word problems with equation templates do not present in the training set. Figure 4.7 shows that, as expected, the mask worked and that we were able to lower the number of syntactically wrong equations compared to the result of the Ultimate template from the previous experiment, which can be seen in Figure 4.3.

Here is an example of a word problem, which is solved by an equation that is not part of the training set:

```
Word problem:
V sadu je 6 řad jabloní s 12 stromy v každé řadě.
Celkem se letos nasbíralo 648 košů jablek.
Kolik košů jablek v průměru obdrží každá jabloň?

Word problem in English:
There are 6 rows of apple trees in the orchard,
with 12 trees in each row.
A total of 648 baskets of apples were collected this year.
How many baskets of apples on average will each apple tree receive?

Word problem template:
v NOUN1 být NUM1 NOUN2 jabloň s NUM2 NOUN3 v každý
NOUN2 . celkem se letos nasbírat NUM3 NOUN4 NOUN5 . kolik NOUN4 NOUN5 v
NOUN6 obdržet každý jabloň ?

Equation:
( NUM3 / NUM1 ) / NUM2

Prediction:
NUM3 / NUM1 / NUM2

Result: 9
```

We may observe an issue from the above example mentioned in Subchapter 3.3. There are several options to evaluate the model's output, such as comparing equations and comparing results. The example above shows that some equations in the training set may contain redundant parentheses, which we can omit and still get the syntactically correct result. For future work, preprocessing should be extended with an equation simplification, simplifying the word problem equation to its elementary form while preserving the core information. We can use a simplification of the equation by the Python library SymPy[1].

Let us compare our model's accuracy with DNS and the solver presented in my bachelor thesis. As we mentioned in Subchapter 1.1, DNS achieved 64.7% on the Math23K dataset. This accuracy is for an approach based on using a seq2seq and retrieval model. Unfortunately, we cannot evaluate our model on the Math23K dataset, and we cannot compare accuracy directly because there are several incomparable facts. We

[1] https://www.sympy.org/en/index.html

do not know how many epochs were used to train the DNS model. It is also worth noting that the best precision mentioned in DNS is not purely based on seq2seq. The achieved accuracy using purely seq2seq model by DNS was 53.7% without using SNI and 58.1% using SNI in the Math23K dataset. The solver from my bachelor thesis was based on representing a word problem as a point in a vector space, and it was trained and evaluated using the SVM algorithm. The precision of this approach was 74.34% for all *reischigova* word problems. Our approach using seq2seq achieved slightly better accuracy - 77.56%. However, we have to take into account that the approach from the bachelor thesis cannot solve a word problem that was not part of the training set, which leaves only 13 unique equations the solver can deal with. The proposed approach can, in theory, handle any equation, including those not present in the dataset.

Chapter 5
Conclusion

This thesis dealt with the problem of solving Czech word problems using neural networks. We performed several experiments using seq2seq architecture as a primary approach for solving word problems and made several observations from these experiments. One of the essential observations made is the importance of preprocessing the word problem and how it is essential to identify critical features of the language to generalise natural input language to a model, such as lemmatisation or how each part of speech contributes to the model. Although this thesis does not present revolutionary seq2seq architecture, it evaluates an existing architecture and presents its improvements. It confirms that general seq2seq architectures can be used for a broader class of natural language tasks, even though their primary usage was focused on something else. In the case of our used architecture, the primary usage was machine translation.

In future work, we suggest focusing on preprocessing intensively as experiments showed that the use of preprocessing leads to significant improvements. Auxiliary tools for preprocessing are essential, such as UDPipe, which is beneficial for natural language preprocessing, and their development is crucial for future research. Unfortunately, we find poor availability of the existing solver's implementations and unsatisfactory standardisation and availability of datasets. It would be enormously beneficial for future work to define a standard for datasets and make the solver's implementations accessible.

The definition of scope should be the first thing that is done before creating a solver to specify what the solver is expected to do. As we mentioned in this thesis, there are several pitfalls that have to be defined before the solver and dataset are created, such as what type of word problems do we want to solve, how do we want to evaluate them, and whether we need step-by-step instructions on how to get equation or solution, and many others.

Using a sequence-to-sequence model, we obtained a general solver for the word problem capable of solving word problems that are not present in the training dataset. We obtained solid results using this technique, but we should not consider this technique as the peak of word problem solvers. We should pay attention to older approaches that tried to model a solver that is capable of producing step-by-step instructions to solve the word problem. These solvers would be more beneficial because they would provide us with information extraction and could be used in other NLP tasks like chatbots or voice assistants.

It is no doubt that solving word problems in Czech is a more complex task than solving word problems in English or Chinese. There are several arguments for this statement. As mentioned before, the Czech language is a fusional language, unlike English and Chinese. There is the inflexion of words which depends on gender, nominative case, or singular number. On the other hand, English and Chinese belong to so-called analytic languages, a group of languages with a strict syntax and do not use inflexion. Another argument supporting the statement above is that there are more datasets and tools for working with these languages. We know what to expect from the language, thanks to strict syntax.

As far as we know, the proposed solver is one of the first solvers for the word problems in the Czech language using the seq2seq approach, which achieved 61.24% using SNI. The proposed solver has the potential to achieve 72.72% on the whole dataset if it identifies all significant numbers correctly. For comparison, the pioneer solver DNS achieved 53.7% without using SNI and 58.1% with using SNI on dataset Math23K. Both achievements are purely for the seq2seq approach. It should be essential to note that DNS was trained and evaluated on a similar size dataset, but for the English language and entirely using the trick, which causes that there are no syntactically incorrect equations. We find our solution to be a significant success for the word problem solver for the Czech language. We want to encourage others to explore word problem solvers for other languages than English and Chinese.

References

[1] Dongxiang Zhang, Lei Wang, Luming Zhang, Bing Tian Dai, and Heng Tao Shen. The gap of semantic parsing: A survey on automatic math word problem solvers. *IEEE transactions on pattern analysis and machine intelligence*. 2019, 42 (9), 2287–2305.

[2] Daniel G. Bobrow. *Natural Language Input for a Computer Problem Solving System*. 1964.

[3] James R Slagle. Experiments with a deductive question-answering program. *Communications of the ACM*. 1965, 8 (12), 792–798.

[4] Charles R Fletcher. Understanding and solving arithmetic word problems: A computer simulation. *Behavior Research Methods, Instruments, & Computers*. 1985, 17 (5), 565–571.

[5] Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. *Learning to Automatically Solve Algebra Word Problems*. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, 2014. 271–281.
`https://www.aclweb.org/anthology/P14-1026`.

[6] Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. *How well do Computers Solve Math Word Problems? Large-Scale Dataset Construction and Evaluation*. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, 2016. 887–896.
`https://aclanthology.org/P16-1084`.

[7] Yan Wang, Xiaojiang Liu, and Shuming Shi. *Deep Neural Solver for Math Word Problems*. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, 2017. 845–854.
`https://aclanthology.org/D17-1088`.

[8] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. *Sequence to sequence learning with neural networks*. In: *Advances in neural information processing systems*. 2014. 3104–3112.
`https://storage.googleapis.com/pub-tools-public-publication-data/pdf/43155.pdf`.

[9] Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. *Translating a Math Word Problem to a Expression Tree*. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, 2018. 1064–1069.
`https://aclanthology.org/D18-1132`.

[10] Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. *Mathdqn: Solving arithmetic word problems via deep reinforcement learning.* In: *Proceedings of the AAAI Conference on Artificial Intelligence.* 2018.

[11] Kadlec Jan. *Automatické vyhodnocování matematických slovních úloh.* Bachelor thesis, Czech Technical University in Prague. 2020.

[12] Oishik Chatterjee, Aashish Waikar, Vishwajeet Kumar, Ganesh Ramakrishnan, and Kavi Arya. *A Weakly Supervised Model for Solving Math word Problems.* 2021.

[13] Jinghui Qin, Xiaodan Liang, Yining Hong, Jianheng Tang, and Liang Lin. *Neural-Symbolic Solver for Math Word Problems with Auxiliary Tasks.* 2021.

[14] Subhro Roy, and Dan Roth. *Unit dependency graph and its application to arithmetic word problem solving.* In: *Proceedings of the AAAI Conference on Artificial Intelligence.* 2017.

[15] Wei Zhao, Mingyue Shang, Yang Liu, Liang Wang, and Jingming Liu. *Ape210K: A Large-Scale and Template-Rich Dataset of Math Word Problems.* 2020.

[16] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. *Program Induction by Rationale Generation: Learning to Solve and Explain Algebraic Word Problems.* In: R Barzilay, and MY Kan, eds. *PROCEEDINGS OF THE 55TH ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS (ACL 2017), VOL 1.* 2017. 158-167. ISBN 978-1-945626-75-3. 55th Annual Meeting of the Association-for-Computational-Linguistics (ACL), Vancouver, CANADA, JUL 30-AUG 04, 2017.

[17] Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. *MAWPS: A math word problem repository.* In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies.* 2016. 1152–1157.

[18] Marie Reischigová. *Matematika na základní a obecné škole ve slovních úlohách.* Pansofia, 1996.

[19] Petr Šulc. *Procvičování - Slovní úlohy pro 2. ročník.* PIEROT s.r.o., 2018. ISBN 978-80-7353-622-0.

[20] Petr Šulc. *Procvičování - slovní úlohy pro 3. ročník.* PIEROT s.r.o., 2018. ISBN 978-80-7353-623-7.

[21] Petr Šulc. *Procvičování - slovní úlohy pro 4. ročník.* PIEROT s.r.o., 2018. ISBN 978-80-7353-624-4.

[22] Petr Šulc. *Procvičování - slovní úlohy pro 5. ročník.* PIEROT s.r.o., 2018. ISBN 978-80-7353-625-1.

[23] *A Neural Network for Machine Translation, at Production Scale.* 2016. `https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html`.

[24] Aditya Mohanty. *SEQ2SEQ model and the exposure bias problem.* 2019. `https://medium.com/analytics-vidhya/seq2seq-model-and-the-exposure-bias-problem-962bb5607097`.

[25] Sepp Hochreiter, and Jürgen Schmidhuber. Long short-term memory. *Neural computation.* 1997, 9 (8), 1735–1780.

[26] Haşim Sak, Andrew Senior, and Françoise Beaufays. *Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition.* 2014.

[27] Aya Abdelsalam Ismail, Timothy Wood, and Héctor Corrada Bravo. *Improving Long-Horizon Forecasts with Expectation-Biased LSTM Networks*. 2018.

[28] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. *On the Properties of Neural Machine Translation: Encoder–Decoder Approaches.* In: *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation.* Doha, Qatar: Association for Computational Linguistics, 2014. 103–111.
`https://aclanthology.org/W14-4012`.

[29] Saddam Abdulwahab. Deep Learning Models for Paraphrases Identification. *MS thesis.* 2017,

[30] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014.

[31] Thang Luong, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation.* In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.* Lisbon, Portugal: Association for Computational Linguistics, 2015. 1412–1421.
`https://aclanthology.org/D15-1166`.

[32] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers).* Minneapolis, Minnesota: Association for Computational Linguistics, 2018. 4171–4186.
`https://aclanthology.org/N19-1423`.

[33] Milan Straka. *UDPipe 2.0 Prototype at CoNLL 2018 UD Shared Task.* In: *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies.* Brussels, Belgium: Association for Computational Linguistics, 2018. 197–207.
`https://www.aclweb.org/anthology/K18-2020`.

[34] Diederik P Kingma, and Jimmy Ba. Adam: A method for stochastic optimization. 2014,

Appendix A
Glossary

BiLSTM	Bidirectional long short-term memory
ConvS2S	Fully convolutional sequence-to-sequence
DNS	Deep Neural Solver
GRU	Gated recurrent unit
LSTM	Long short-term memory
NLP	Natural language processing
ReLU	Rectified Linear Unit
RNN	Recurrent neural network
SGD	Stochastic gradient descent

Appendix B

Distribution of equations in the data set

equation	count
NUM1 * NUM2	2984
NUM1 / NUM2	1377
NUM1 * NUM2 * NUM3	1342
NUM1 + NUM2	1258
NUM1 * NUM2 / NUM3	1094
NUM1 - NUM2	956
NUM1 * NUM2 + NUM3	921
NUM1 * NUM3 / NUM2	695
NUM2 / NUM1	672
(NUM1 - NUM2) / NUM3	663
NUM2 * NUM3 / NUM1	604
NUM3 * (NUM1 + NUM2)	569
(NUM1 - NUM3) / NUM2	529
NUM1 - NUM2 + NUM3	475
(NUM1 + NUM2) / NUM3	466
NUM1 / (NUM2 * NUM3)	453
- NUM1 + NUM2	367
NUM1 * (NUM2 + NUM3)	361
NUM1 - NUM2 * NUM3	359
NUM1 - NUM2 - NUM3	354
NUM3 / (NUM1 * NUM2)	321
NUM1 + NUM2 - NUM3	318
NUM1 * NUM2 - NUM3	316
NUM1 + NUM2 + NUM3	278
(NUM1 - NUM2 * NUM3) / NUM4	228
NUM1 + NUM2 * NUM3	199
NUM1 / NUM2 - NUM3	170
- NUM1 + NUM2 + NUM3	167
NUM1 / (NUM2 + NUM3)	166
(NUM2 - NUM3) / NUM1	165

Table B.1. The figure shows the distribution of equation templates in the created dataset. There are 782 unique equation templates in the dataset after equation simplification using Python library *SymPy*. It is worth noting that these equations respect number order in the word problem. We plot only 30 of them, which have a count higher than 150.

Appendix C
Technical documentation of the source code

Automatic Evaluation of Mathematical Word Problems

This source code was created for the diploma thesis of the same name. As the name suggests, the goal is to evaluate mathematical word problems automatically. Word problems in the Czech language for pupils of primary schools are considered. The implemented method is based on the seq2seq model and template preprocessing.

Required libraries to run the program are specified in file requirements.txt. The best practice is to create a virtual environment first and install requirements there. The procedure for creating and using a virtual environment is listed below. If you would like to train your model, the recommendation for faster learning is to use GPU, if possible.

```
# This command creates a virtual environment named .venv
root@computer:~$ python3 -m venv .venv

# Activate virtual environment
root@computer:~$ source .venv/bin/active
# After virtual environment activation, you should see the following line
(.venv) root@computer:~$

# Now install the requirements.txt file
(.venv) root@computer:~$ pip3 install -r requirements.txt
```

> ***IMPORTANT:*** If you want the solver demo to work, you need to download the model for UDPipe. The following link will download czech-pdt-ud-2.5-191206.udpipe directly or you can find it in Universal Dependencies repository. Placed download file to the model directory and do not rename it.

Project structure

```
.
├── model
│   ├── data
│   │   ├── __init__.py
│   │   ├── base.py
│   │   ├── check.py
│   │   ├── dataset.py
│   │   ├── template.py
│   │   └── word_problem.py
│   ├── utils
│   │   ├── __init__.py
│   │   ├── file_utils.py
│   │   ├── help_functions.py
│   │   ├── model_actions.py
│   │   ├── model_constants.py
│   │   └── model_utils.py
│   ├── __init__.py
│   ├── conllu_record.py
│   ├── decoder.py
│   ├── encoder.py
│   ├── seq2seq_wrapper.py
│   ├── sequence.py
│   ├── udpipe_model.py
│   └── word2number.py
├── trained_models
│   ├── sni
│   │   └── UltimateSni_model.pt
│   └── wp_solvers
│       ├── UltimateTemplate_model.pt
│       └── experiment_UltimateTemplate_model.pt
├── README.md
├── evaluate.py
├── requirements.txt
├── sni_train.py
├── templates_introduction.ipynb
└── train.py
```

Templates

- NoSubTemplate
- SimpleTemplate
- SmarterTemplate
- UltimateTemplate

All templates are located in file template.py.

Train

Train word problem solver model

Word problem solver can be trained using scripts in file train.py using functions `run` or `run_existing`. For the usage, read the description of the functions.

Train significant number identification model

SimpleTemplate, SmarterTemplate, and UltimateTemplate require significant number identification to define the mapping for number constants when the mapping can not be determined deterministically. We can train a model which determines which numbers are significant (are part of the equation).

The training can be performed in file sni_train.py using functions `run` or `run_existing`. For the usage, read the description of the functions.

Run script

When you train a model, you can easily create an application which can evaluate it using the file evaluate.py.

The following script runs a solver on the default word problem.

```
Joan našla na pláži 70 mušlí. Dala Samovi několik mušlí. Má 27 mušlí. Kolik
mušlí dala Sam?
```

using experiment_UltimateTemplate located in trained_models/wp_solvers.

```
python3 evaluate.py --example
```

Using argument `--word_problem`, you can specify your word problem.

```
python3 evaluate.py --word_problem "Petr má tři jablka, Pavel má dvě
jablka a Eva má čtyři jablka. Kolik jablek mají dohromady?"
```

The following command creates a loop asking the user for word problem input, outputting the predicted equation. You can quit the program by typing 'q'. If you do not specify the model, the default model will be used.

```
python3 evaluate.py
```

Using the argument `--model`, you can specify which model will be used. If nothing else is specified, it creates a loop waiting for the input.

```
python3 evaluate.py --model UltimateTemplate
```

Run notebook

We provide a Jupyter notebook introducing templates. The Jupyter notebook contains text with commands showing the user how the preprocessing affects the word problem.

Appendix D
Documentation of the dataset

Basic information about dataset

This dataset contains two partly translated existing datasets - Ape210K, arithmetic. Unfortunately, we cannot provide word problems from schoolbooks and collections due to legal reasons.

Directory structure

```
├── Ape210K
│   ├── README.md
│   └── final.json
├── README.md
└── arithmetic
    ├── README.md
    └── final.json
```

Both directories contain a final.json file containing word problems and a README.md file with information about the original dataset and translated dataset.

How to get access to the whole dataset or how to contribute

Suppose you are interested in these word problems and creating a word problems dataset for the Czech language. Please, reach out Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague or directly doc. RNDr. Daniel Průša, Ph.D. who has access to the entire dataset. You are more than welcome to contribute.

An example of word problem from dataset:

```
{
    "word_problem_CZ": "Joan našla na pláži 70 mušlí. Dala Samovi několik mušlí. Má 27 mušlí. Kolik mušlí dala Sam?",
    "equation_CZ": [
        "70 - 27"
    ],
    "result": [
        43
    ],
    "secret": false,
    "source": "arithmetic",
    "word_problem_original": "Joan found 70.0 seashells on the beach . She gave Sam some of her seashells . She has 27.0 seashells . How many seashells did she give to Sam ?",
    "equation_original": [
        "70.0-27.0"
    ]
}
```