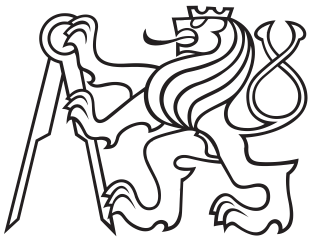
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

NLP Methods for Word Problems

Bc. Ronald Krist

Supervisor: Ing. Jan Drchal, Ph.D.
Field of study: Open Informatics
Subfield: Artificial Intelligence
May 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Krist** Jméno: **Ronald** Osobní číslo: **434780**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Metody zpracování přirozeného jazyka pro řešení slovních úloh

Název diplomové práce anglicky:

NLP Methods for Word Problems

Pokyny pro vypracování:

Seznam doporučené literatury:

[1] Wang, Lei, et al. "Mathdqn: Solving arithmetic word problems via deep reinforcement learning." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 32. No. 1. 2018.
[2] Zhang, Dongxiang, et al. "The gap of semantic parsing: A survey on automatic math word problem solvers." IEEE transactions on pattern analysis and machine intelligence (2019).
[3] Wang, Lei, et al. "Translating a math word problem to an expression tree." arXiv preprint arXiv:1811.05632 (2018).
[4] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Jan Drchal, Ph.D. centrum umělé inteligence FEL

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

Datum zadání diplomové práce: **21.02.2021** Termín odevzdání diplomové práce: **20.05.2022**

Platnost zadání diplomové práce: **19.02.2023**

Ing. Jan Drchal, 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

Acknowledgements

I would like to thank the supervisor of this thesis, Ing. Jan Drchal, Ph.D. for always being nice and helpful.I'm also thankful for the support of my family and the moral support of my friends and my cat and spotify playlists, which possibly kept me from going insane.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on 16. May 2022

Abstract

This work explores methods for adapting existing approaches of automatic solving of math word problems into the Czech language. It explores the use of machine translation to overcome data limitations. It evaluates the results on two models based on recurrent neural networks and the transformer architecture, including employment of pre-trained language models.

Keywords: word problems, natural language processing, transformers, machine translation

Supervisor: Ing. Jan Drchal, Ph.D.

Abstrakt

Tato práce zkoumá způsoby adaptace existujících postupů automatického řešení slovních úloh pro úlohy v Českém jazyce. K překonání nedostatku dat zkouší využití automatického překladu. Výsledky vyhodnocuje na dvou modelech založených na rekurentních neuronových sítích a transformerech. Zkouší také použití předtrénovaných jazykových modelů.

Klíčová slova: slovní úlohy, zpracování přirozeného jazyka, transformers, strojový překlad

Překlad názvu: Metody zpracování přirozeného jazyka pro řešení slovních úloh

Contents

Figures

Tables

Chapter 1

Introduction

The ability to work with numbers and to reason with quantities, is necessary for any deeper reasoning about our reality, including reasoning about text in natural language. To get meaningful information out of a text like *I woke up at 11 and went to work one hour later*, we need to recognize that the strings "11" and "one" represent a number, parse them into their respective numeric values, deduce that the number 11 represents time of the day and 1 represents additive quantity relative to it, add the numbers to get the time of departure to work and infer from common sense that it's probably quite late [TPSI21].

Natural language processing (NLP) systems rarely give special consideration to numbers and approach them in a fashion similar to other words, which contrasts with the consensus in neuroscience suggesting that numbers in the brain are represented differently.

[TPSI21] suggests a taxonomy of tasks for researchers working on improving the numerical reasoning capabilities of NLP systems. They divide the tasks into four types. This division is based on granularity - whether they work with exact or approximate quantities - and units - whether they work with abstract or grounded units. They come up with seven types of tasks: simple arithmetic - the task of doing arithmetic operations over numbers alone, numeration - decoding numbers from strings, magnitude comparison, learning exact common sense numeric facts, like that lion has four legs, estimation of common sense

quantities, eg. how tall are spruces, numeric language modeling and math word problems.

The last type, math word problems, is recently enjoying increasing attention, owing to the growing popularity of employing deep learning in NLP tasks. A math word problem is a short narrative in natural language presenting some state of the world including numeric quantities and a question asking to deduce some unknown quantity from the description. Such problems are commonly solved during elementary school education. Development of an automatic computer solver is challenging due to the difficulty in parsing natural language into machine-understandable logic. Hence, a system that reliably solves math word problems could prove decent ability of quantitative reasoning in natural language and be a milestone toward general AI [ZWZ+19].

One interesting research direction lies in probing how the ability of current solvers is affected by language. For example, how well will they work in languages with more relaxed word order, like the Czech language [KP20]. This thesis aims to explore the popular state-of-the-art solvers and datasets and assess the performance on data in the Czech language.

Chapter 2

Background

This chapter presents the theoretical background from which our work stems. The chapter is divided into five sections:

- The first section describes the goal of our work.
- The second section gives an overview of the development of attempts to solve the problem, ending with the current state-of-the-art.
- The third section presents to our awareness the only preceding attempt to solve Czech math word problems.
- The fourth section provides an analysis of data used for training and evaluation of the state-of-the-art solvers.
- The last section provides an overview and comparison of the performance of the state-of-the-art solvers.

2.1 Problem description

The typical math word problem (MWP) is a short narrative in natural language describing partial state of the world, including some explicitly stated

Body	”There are fifty-eight students trying out for the school’s trivia teams. If twenty-eight of them didn’t get picked for the team and the rest were put into five groups,”
Question	”how many students would be in each group?”
Quantities	58, 28, 5
Expression	(58-28)/5
Solution	6

Table 2.1: Example of an arithmetic word problem.

quantities, and poses some queries about unknown quantities - variables.

There are several types of MWPs depending on the type and number of operations required to compute the queried quantities. Our work focuses only on a subset of MWPs, arithmetic word problems, which are problems with one unknown quantity that can be computed using basic arithmetic operators. The other, more complex MWP types include problems requiring a set of equations or more complex mathematical operations.

Arithmetic word problems (AWPs) query for a single variable, which can be expressed by an equation made from numbers, four fundamental operators and priority brackets, $O = (,), +, -, *, /$.

Since the problem body - the narrative describing state of the world, the problem question and numeric quantities all form a logically different part of the problem, we choose to represent them independently. Formally, the problem input I is then represented as two sequences of k_b and k_q words $\left\langle w_{b_0}, w_{b_1}, .., w_{b_{k_b}} \right\rangle$ and $\left\langle w_{q_0}, w_{q_1}, .., w_{q_{k_q}} \right\rangle$ and n quantities mentioned in the text, $q_1, q_2, .., q_n$.

The goal is to detect and resolve some unknown quantity x mentioned in the text. This is usually done by mapping the problem input I into an arithmetic expression E, which can be evaluated to provide the unknown quantity x.
We can think of the solver as a mapping function $S : I- > E$.

An example of an AWP is provided in table

2.1.1 Properties of MWPs

The difficulty of various MWPs depends on the various properties of the problem. It is beneficial for researchers to be aware of the types of the problems, as it can help them detect directions for improving their solvers.

One of the basic properties of an arithmetic word problem is whether it is a single-step or multi-step problem. Single-step problems require just one mathematical operation, while multi-step two or more. Historically, datasets often contained only single-step problems, however, all commonly used modern benchmark data include multi-step problems too.
Another solver-relevant property is whether the problem contains irrelevant information, typically some irrelevant quantity or "red-herring" sentences, sentences that provide no useful information for the derivation of the solution.

Problems can also contain background knowledge, which is information that is assumed the solver should know, therefore not explicitly stated in the problem text. Typically it is common sense knowledge or specific domain knowledge. For example, the value of pi or that dog has four legs. Modern datasets usually have few problems with this property. For the purposes of this thesis, we will not work with these problems.

One way to further categorize the problems is according to the school grade level in which they are typically solved. This should provide basic insight into the difficulty of the problem. Most datasets don't label their examples with this information, ASDiv [MLS21] categorizes them ranging from grade 1 to grade 6.

A different attribute is the primary mathematical procedure or "thought pattern" required to solve the MWP. This doesn't mean only the mathematical operations in the expression but also the thought patterns behind them, therefore two problems with the same target expression can be categorized differently. There isn't a unique, clear taxonomy and the categorization is mostly left to each author's insight. ASdiv [MLS21] labels problems with their own categorization system. The details of their categorization will be provided later in our work.

2.2 Solver development

This chapter reviews methods developed for automatic solving of math word problems. The history of designing automatic solvers for MWPs goes back to the 1960s, when the first solver, STUDENT, was developed [ZWZ+19]. Historically, [ZWZ+19] suggests that three major directions were taken to address the problem. The oldest one, which was dominant up to around the year 2010, is based on rule matching methods. Methods based on semantic parsing and statistical machine learning became popular after, and deep neural learning approaches gained traction recently.
Automatic MWP solving is an active research field. Solvers capable of solving most AWPs in general, without additional constraints on the input or expression format, were developed only recently. And even these still can't solve specific types of AWPs, specifically problems with implicit quantities and background knowledge, though the latter is starting to be addressed too.

2.2.1 First phase

In the first pioneering stage, roughly from the year 1960 to 2010, systems such as STUDENT, DEDUCOM, WORDPRO and ROBUST depend on manually crafted rules for matching patterns in the input text. The input usually must be preprocessed into some specific format on which the solvers can operate.
For example, WORDPRO takes the problem in the form of sequential propositions.
CHIPS and ARITHPRO solve problems in which owner of some objects transfers them to their receiver. The first input sentence must be the number of objects the owner has and the second sentence must contain the word "gave" to signalize the operation of transfer of the owner's objects to their receiver.
The most recent solver ROBUST is capable to solve multi-step problems.
The main drawback of these solvers is their reliance on specific input formats and rigid rule sets, therefore they heavily rely on human rule handcrafting and can only resolve a limited number of scenarios defined in advance. This approach is only historical now [ZWZ+19][MG08].

2.2.2 Second phase

Since the 2010s, researchers started to incorporate the power of machine learning into their works. The base of these approaches is to use semantic parsing to map the sentences of the MWP into some structured logic representations and then use these representations to make the final inference. To improve their performances, the developers employed various techniques from feature engineering and statistical machine learning.
The usual approach is to predefine some internal problem representation templates and learn classifiers to identify the required entities, quantities, and operators to fill the templates and yield the resulting expression using a simple logical procedure [ZWZ+19].

[RVR15] uses a simple template consisting of two variables and an operator to solve single-step AWPs. It employs three classifiers, one to detect two relevant quantities in the problem text, one to infer which operator to use, and the last one for the order of the quantities, if the operation isn't commutative.

More complex approaches are needed for multi-step reasoning. One such example is ARIS [SK15], which defines a logic template called schema, containing a set of entities, their containers, attributes, quantities and relations. The text is split into fragments using hand-crafted rules, which work on POS tagging, dependency parsing, and coreference resolution of the problem text. Each fragment contains one entity with associated quantity and attributes inside one or two containers and one verb. Quantities in the entities are then updated according to a classification of the fragment verb. The verbs are categorized into seven classes and an SVM classifier is trained to predict them during inference. The feature vector of the classifier is composed from the similarity of the classified verb to a set of seed verbs, categories of the classified verb in WordNet, the frequencies with which the verb appeared in each category in the WordNet reference corpus, and the dependency relations with other words in the fragment.

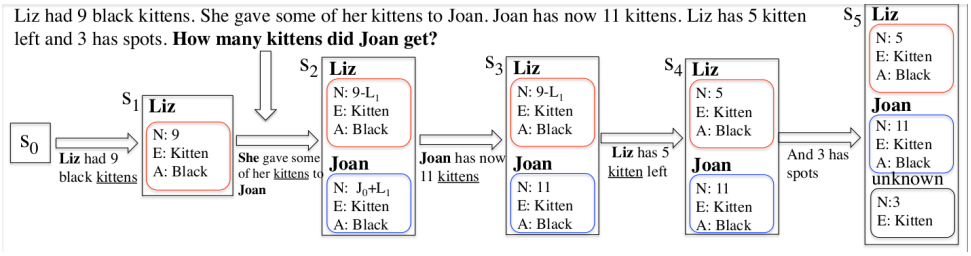


Figure 2.1: Process of updating the world states in ARIS, reprinted from [SK15].

In general, these methods have two important drawbacks. Firstly, the internal templates and classifier features have to be handcrafted, which complicates adaption to new circumstances, like adding a new operator.
For example, ARIS is developed to work only on addition and subtraction problems. Extending it to multiplication and division would require determining additional verb categories with no guarantee of satisfactory results. Differently, a new dataset could feature significant amount of problems with different logical structures or "thought patterns" required to solve them. An approach tailored for the original data may not work very well on them.
The second problem is that most of these methods require additional training data annotations, like the verb categories in case in ARIS.
A positive aspect compared to the deep learning methods is that it's still sometimes possible to see steps used in answer derivation, for example in the development of the world states by ARIS.

An important direction in development first considered during this phase rests on the fact that arithmetic expressions can be expressed as binary trees. Such tree has quantities as leaf nodes and operators as inner nodes, going from lowest priority operators in the top parts of the tree to the highest priority at the bottom. See figure. The idea is to transform the derivation of the expression into constructing the equation tree. From high level perspective, the usual procedure starts with detecting possibly relevant quantities in the problem text. Relevant quantity means a quantity which appears in the solution expression. Then the possible candidate trees are enumerated. Finally classifiers are used to select a tree with the highest likelihood of providing the correct expression. The classification typically uses sum of local classifiers determining the likelihood of connecting two nodes with their parent operator and a global classifier based on the features of the whole tree. Using this approach, developers avoid both additional data annotations and specific templates.

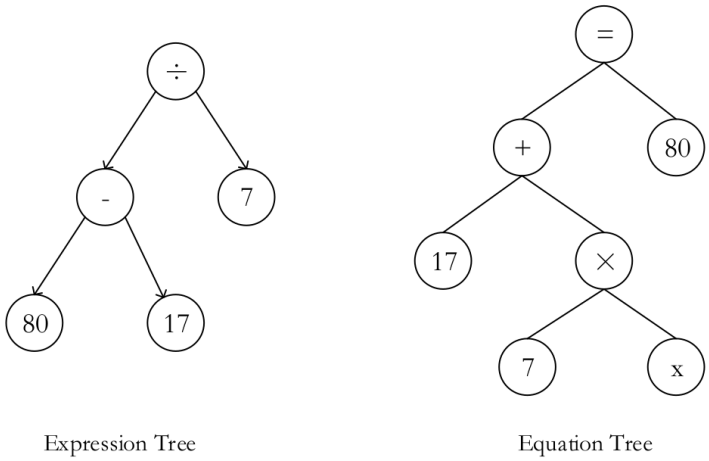


Figure 2.2: Two variants of trees representing an arithmetic expression, reprinted from [ZWZ+19].

As mentioned before, these models strongly rely on handcrafted features used in their classifiers. The selection should reflect properties of the text thought to be necessary for solution. As such, they deserve special mention. Table shows commonly used features during the second phase.

Feature	Type	Example
Quantity refers to rate?	Quantity	”Each ride costs 5 tickets” - 5 is related to rate
Is between 0-1	Quantity	
Is 1 or 2?	Quantity	
Context Lemmas	Context	“Connie has 41.0 red markers.” (around 41.0, window size 4) -> Connie, have, red, marker.
Context POS tags	Context	“A chef needs to cook 16.0 potatoes.” (around 16.0, window size 2) -> TO, VB, NNS.
Context Dependencies	Context	“Ned bought 14.0 boxes of chocolates candy.” (root bought, window size 2) -> (boxes, 14.0) → (num), (boxes, of)→ (prep), (bought, Ned) → (nsubj)
Comparative adverbs	Context	For “If she drank 25 of them and then bought 30 more.” , “more” is a comparative term in the window of quantity “30” .
Same unit?	Quantity-pair	“Student tickets cost 4 dollars and general admission tickets cost 6 dollars” ; 4 and 6 have same unit.
If quantity associated with rate, is the second quantity associated with its unit?	Quantity-pair	For “each box has 9 pieces” and “Paul bought 6 boxes of chocolate candy”, “9”is related to a rate (i.e., pieces/box) and “6” is associated to the unit “box” .
Appear in the same sentence?	Quantity-pair	
First quantity > other?	Quantity-pair	
Unit or related noun phrase of quantity in question?	Question	
Unit or related noun phrase of quantity has the highest number of match tokens with the question	Question	For the question “How many apples are left in the box?” and a quantity 77 that appears in “77 apples in a box” , there are two matching tokens (” apples” and “box”)
Number of quantities which happen to have the maximum number of matching tokens with the question	Question	”Rose have 9 apples and 12 erasers. ... 3 friends. How many apples dose each friend get?”, the number of matching tokens for quantities 9, 12 and 3 is 1, 0 and 1. There are two quantities with the maximum matching token number.
Any component of the rate is present in the question?	Question	“How many blocks does George have?” and a quantity 6 associated with rate “blocks/box” , the feature indicator is set to 1 since block appears in the question.
Terms like ”each” or ”per” in question	Question	
Comparison terms like ”more” or ”less” in question?	Question	
”how many” like terms in question	Question	
Dependent verb of quantity	Verb	
Distance vector between verb and few seed verbs	verb	
Two quantities have same dependent verb?	verb	“In the first round she scored 40 points and in the second round she scored 50 points” , “40” and “50” both have the same verb “scored” .
Both dependent verbs refer to the same verb mention?	verb	“She baked 4 cupcakes and 29 cookies.” , “4” and “29” both shared the verb “baked” .
Number of quantities in the text	global	

Table 2.2: Common features used in MWP solvers, based on [ZWZ+19].

In our opinion, for the general research in artificial intelligence, perhaps the biggest drawback of these methods is their reliance on rigid manual feature engineering. The solvers are limited by the quality of the feature design, while the deep learning models are allowed to distill the features from the data on their own. This amounts to a higher level of abstraction and transferability to different domains and tasks.

2.2.3 Neural NLP

Following broader trends in modern machine learning, researchers recently started to leverage the power of deep learning to learn features automatically with minimal handcrafting, therefore providing more robust and generalizable performance. They constitute the state of the art of MWP solving. This section follows the development of modern deep learning-powered NLP from the perspective of MWP solvers.

General architecture

The types of deep neural models used to solve MWPs are usually the seq2seq models. On a high level, they take a sequence as input and transform it into a different sequence on output.

Typically, seq2seq models are based on encoder-decoder architecture.
They use two neural networks, an encoder and a decoder. The encoder creates vector representation (embedding) of the input. This vector embedding is supposed to provide encoded summarization of the meaning of the input sequence. The embedding is then passed to the decoder, which produces the desired solution step by step. This approach allows variable input and output sequence lengths [ZDLS20].

Recurrent neural networks and attention

Until recently, recurrent neural nets were typically used in state-of-the-art seq2seq models. Recently, the use of transformer-based architectures is becoming dominant [ZDLS20]. State-of-the-art MWP solvers still feature RNNs sometimes.

Conceptually RNNs work by taking the first element of the sequence, passing it into the network, and getting output. Then the second sentence element is passed along with the previous hidden state together, producing the second output. The process is repeated until all elements in the sequence are processed. Due to the sequential nature of this process, the last output should retain some information from the whole sequence.
Typical RNN-based architecture for seq2seq modeling employs RNNs for both the encoder and decoder. The encoder's last output vector, sometimes called the embedding vector, is used by RNN decoder, which produces the output sequence step by step until a specific end of sequence token is generated.

Vanilla RNNs have problems with long-term dependencies. This means that the information from the first elements of the sequence tends to get lost in the final embedding. These distant elements are called long-term dependencies [Hoc98]. To address this issue, specialized network architectures were developed. One of the most used are LSTMs (Long Short Term Memory Units)[HS97]. Along with the output they also pass memory data and allow the network to explicitly control how the memory should be updated and added to the output. They use three types of gates, forget gate, input gate, and output gate. The forget gate allows to partially delete (multiply it with value between (0-1)) the memory data. The input gate allows to add part of the current output to the memory and the output gate adds part of the memory to the output.

The attention mechanism [BCB14] attempts to further reduce this problem. The idea is instead of using just the encoder embedding vector, save the encoder's hidden states for each element in the input sequence. The hidden states are then along with the current decoder output fed into a specialized neural network. This network has a softmax layer in the end, producing a vector of weights associated with the encoder's hidden states. This weight vector signifies the importance ("attention") with which the decoder should regard ("attend")

the encoder hidden states in output generation. The Sum of the encoder's hidden states weighted by the weights vector is called the "context vector" and is used along with the base decoder output to produce the final output.

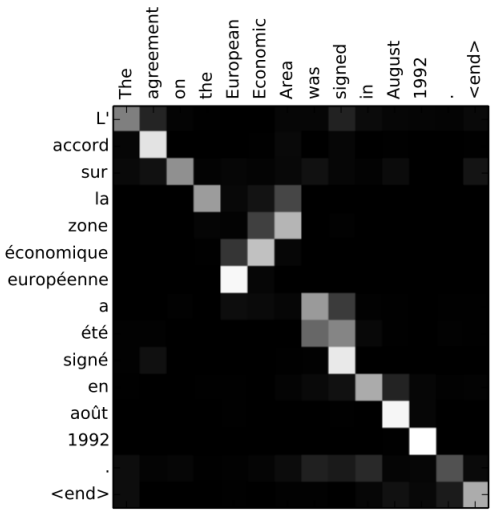


Figure 2.3: Attention weights during machine translation, reprinted from [BCB14].

Despite this effort, RNNs still have their limitations. Their sequential nature prohibits parallelization within training examples [VSP+17]. The next idea was to stop using them completely and instead use an architecture based only on a version of the attention mechanism called self-attention. Self-attention is using the attention mechanism to compute attention of each element in a sequence to each other. This was proposed in an influential paper with a cute title Attention Is All You Need [VSP+17], introducing the Transformer network model.

Transformer

The original transformer model replaces the sequential processing of RNNs with blocks of self-attention layers.

The self-attention in the Transformer uses three sets of vectors called Queries, Keys and Values. The general idea is to construct the weight vector w by eval-

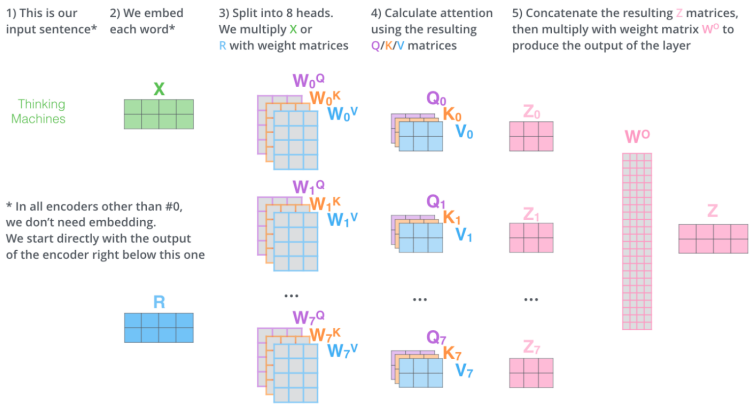


Figure 2.4: Illustration of the multi-head dot scaled dot product self-attention used in Transformer, from [Ala]

uating the similarity between Queries and Keys. The context vector is then computed as the sum of Values weighted by w. The self in self-attention means that the same input is used to construct the Queries, Keys and Values.

More technically, each layer has input dimension $max\_sequence\_length$ x d_{model}. d_{model} is the dimension of the token representations. Three learnable linear transformations then project this input into the Queries, Keys and Values sets. First, we compute the dot product between each vector in Keys and Queries and softmax the results. The resulting weight vectors then represent the attention on each value in Values. New token representation is then acquired by summing the values weighted by the weight vector.
The transformer uses multi-head attention. The idea is to have h differently parametrized, parallel attention processes. This is achieved by lowering the "embedding" dimension of the projections and then concatenating the results to return to the input dimension. According to the authors, this facilitates the ability to look for patterns in different projection spaces. Each layer of the encoder has a sublayer of these self-attention blocks connected with a two-layer feedforward network. The encoder is then just N of these layers connected.
The decoder block additionally has the same attention layer, where the queries and keys are outputs from the last block of the encoder. The decoder is then analogically constructed by this decoder blocks stacked after each other.

Language models

Language models are systems trained primarily on token prediction tasks, ie. predicting the likelihood of some token given its preceding or surrounding context. They have gone through rapid evolution in recent years, going from n-gram models through RNN-powered ones best known for producing word embeddings with the attempt to map their semantic relationships into vector space, finally to transformer-based large pre-trained networks providing the backbone to many downstream tasks [BGMMS21].
Modern NLP is powered by large (featuring billions of parameters and gigantic training datasets) pre-trained models typically based on a transformer-like architecture. These models are meant to provide a general understanding of natural language and as such provide the base for finetuning for specialized downstream tasks. For this reason, they are being increasingly employed in MWP solving [LZSZ21],[PBG21],[SYL+21].

One of the most influential examples is Google's BERT (Bidirectional Encoder Representations from Transformers) [DCLT18].
BERT is trained on unlabeled data with two tasks. The first task is called masked language modeling. BERT is presented with a sequence of tokens, some of them randomly replaced by a special token <MASK>. The goal is to correctly predict the masked tokens. The second task is called next sentence prediction. The model is presented with two sentences and asked to decide whether the second sentence follows the first. Training data contains text with about 800 million words from BooksCorpus and 2.5 billons words from English Wikipedia [DCLT18].
Other well-known examples include the GPT series or BART.

2.2.4 Third Phase

The idea which started the use of deep learning in MWP solving was to use the seq2seq architecture to "translate" the problem into the equation directly. The pioneering solver DNS (Deep Neural Solver) came out in 2017 [WLS17]. In the first step, the DNS creates a mapping to replace the numbers in the text with a list of number tokens. This is done to reduce the output space and to

avoid generating irrelevant numbers. For example, the following MWP:
"Robin was making baggies of cookies with 6 cookies in each bag. If she had 23 chocolate cookies and 25 oatmeal cookies, how many baggies could she make?" will be mapped to: "Robin was making baggies of cookies with num0 cookies in each bag. If she had num1 chocolate cookies and num2 oatmeal cookies, how many baggies could she make?"
The solution expression is then $(num1 + num2)/num0$. These equations with numbers substituted by tokens num<order of the number in the problem text> are called equation templates.

The idea caught on and is continuously being refined in new works. The early challenges the seq2seq approach presented were the production of syntactically invalid expressions and the large space for duplicate equation templates. This was subsequently addressed by using implicit or explicit tree structures[ZWZ+19]. [GK21] resolved this ambiguity by transforming expressions into postfix notation, which led to substantial performance improvement.

An influential idea was presented first in GTS (Goal-driven tree-structured MWP solver) [XS19]. The idea is to use a decoder capable of iteratively producing an expression tree. Authors of GTS present the idea as mimicking the human decision process, in which the solver identifies top layer goals and recursively decides if the goals are finished or depend on the solutions of further subgoals.
For illustration, the goal deconstruction of the above presented MWP is presented below.

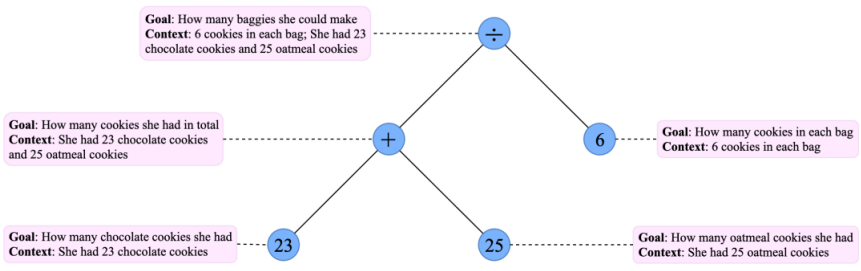


Figure 2.5: Example of the idea of deconstructing MWPs into subgoals, reprinted from [XS19].

The decoder contains three neural modules, the predictor, generator, and

merger. The general process starts by using the encoder embedding as the top goal vector. From this vector, the predictor predicts the tree node token. It can be either an operator, a predefined numeric constant, or a quantity from the problem text. If the token is an operator, the generator creates two subgoals and a new goal vector for the left subgoal from the current goal vector and the predicted token. This process continues iteratively in a depth-first fashion until a leaf node (quantity) is reached. Then the process backtracks and the merger creates subtree embeddings while revisiting parent nodes. The generator then uses these subtree embeddings along with the parent goal vector and token embedding to create the right subgoal.

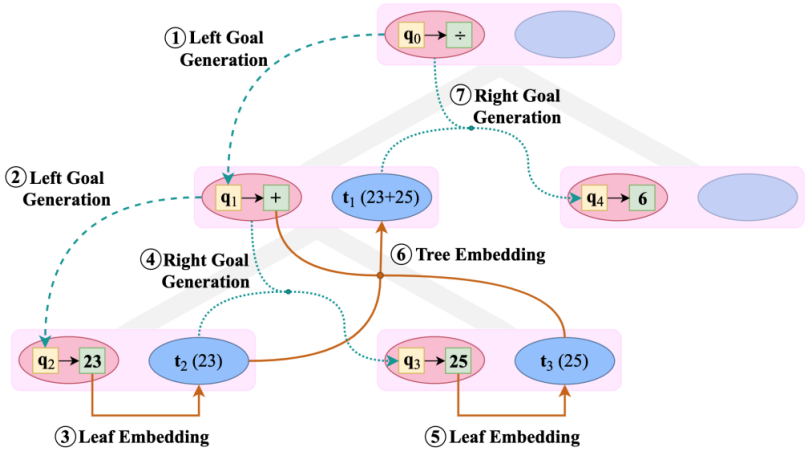


Figure 2.6: Schema of the GTS tree-based decoding process, reprinted from [XS19].

Further refinements of the GTS differ mainly in the neural architectures used for the individual steps, most notably for the encoder. GTS uses bidirectional LSTM, next work Graph2Tree [ZWL+20] preprocesses the problem text into graph representations. The idea is that the graph representation could express the relationships between quantities and other words better than the linear text sequence. The graph also provides information about which quantity is larger. This graph is then fed to a transformer-inspired network to produce a graph embedding used as the first input of the decoder. The latest iteration Mwp-bert [LZSZ21] uses the pre-trained BERT model as the encoder.

[PBG21] studied GTS and graph2tree in adversarial context with the conclusion that they rely on shallow heuristics exploiting artifacts in the datasets (MAWPS

and ASDiv-a).
They found that the models performed well (up to 78% accuracy on MAWPS) even when the question parts of the problems were removed in the test set. This points both to the presence of artifacts in the datasets and that the models exploit them. Evidence for the latter comes from leaving the questions intact in the training set, therefore the solvers aren't forced to exploit them.
Similarly, they found that a model with no word-order information can perform well too. They constructed a model with a simple feed-forward neural network serving as the encoder. The first input to the LSTM decoder with attention is just the sum of hidden states of the last layer of the feedforward network, which can't provide any word order information. According to the authors, this indicates that most of the problems in the datasets can be solved just by associating the occurrence of specific words to their corresponding equations.
Further, they analyzed the activation of attention weights assigned to the hidden representations of input tokens in the bag-of-words model. This could explain its predictions because the hidden states of the feed-forward encoder carry no context information from the rest of the problem. They found that the model usually attends to just a single word no matter the context. This suggests that the models are easily "hackable". See Figure 2.7 for examples.

Input Problem	Predicted Equation	Answer
John delivered 3 letters at every house. If he delivered for 8 houses, how many letters did John deliver?	3 * 8	24 ✓
John delivered 3 letters at every house. He delivered 24 letters in all. How many houses did John visit to deliver letters?	3 * 24	72 ✗
Sam made 8 dollars mowing lawns over the Summer. He charged 2 bucks for each lawn. How many lawns did he mow?	8 / 2	4 ✓
Sam mowed 4 lawns over the Summer. If he charged 2 bucks for each lawn, how much did he earn?	4 / 2	2 ✗
10 apples were in the box. 6 are red and the rest are green. how many green apples are in the box?	10 - 6	4 ✓
10 apples were in the box. Each apple is either red or green. 6 apples are red. how many green apples are in the box?	10 / 6	1.67 ✗

Figure 2.7: Specific words to which the most attention was paid by the constrained model with no word order information, reprinted from [PBG21].

A different approach was presented in Generate & Rank [SYL+21]. The general idea is to use a seq2seq model, called the generator, to generate the K best solutions. Then another model, called ranker, is trained to pick the best solution.
Generate & Rank uses state-of-the-art seq2seq transformer language model, BART, and fine-tune it for generating the ground truth expressions. Then they use the fine-tuned BART to produce K best solutions using beam search. Another L expressions are produced by mutating the solution expression tree. Both of these expression sets form an "expression bank" for one problem. The expressions with the same numeric evaluation are considered positive examples

and the others negatives. The ranker is the same BART model with multi-layer perceptron over the last target token for binary classification score.
Using the ranker provides moderate increase in accuracy compared to just the generator (4-5% for Math23k and MAWPS). The authors especially note the performance increase on problems with long target expressions.

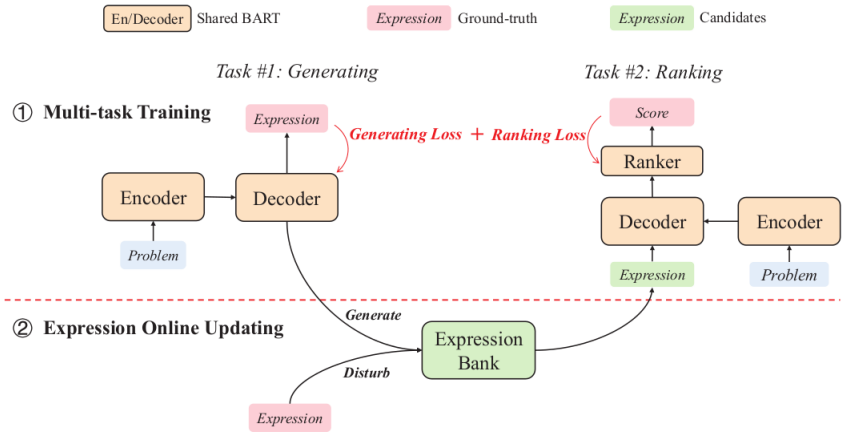


Figure 2.8: Schema of the generate-rank approach, from [SYL+21].

2.3 Word problem solving in Czech

To our awareness, only one attempt at solving MWPs in Czech was developed. For their bachelor's thesis, [KP20] developed a solver which could be placed in the second phase of MWP solvers development as mentioned above.

The solver works as a multi-class classifier, mapping the problem into one of the ten predefined equation templates.
It uses a two-step procedure.

The first step is based on the pattern matching approach of the first phase solvers, however, the patterns are learned from data instead of being handcrafted. The problem texts are lemmatized and adjectives, nouns, and quantities are replaced by their word classes. An algorithm then finds the longest common subsequence between each pair of converted problem texts with the

same equation template. The subsequences are saved, enumerated and the most commonly occurring ones are saved. Subsequences occurring in more problem classes are discarded. This procedure is done for the problem bodies and questions independently.

The second step uses an SVM classifier. The dimensionality of its feature vector is equal to the number of equation templates. The features are computed in the following way:
We take the lemma of the first word of the problem. We look at how many times the word occurred in the bodies of the problems with the same equation template. This number is then weighted by a function assigning weight to each word class. This procedure is repeated for all words in the input problem body and the results are summed. The same is done for the problem question. These two values summed form the first feature. The remaining features are computed analogically.
The word class weighting function is trained empirically using a genetic algorithm with the performance of the SVM as the objective.

The inference follows the steps:
The solver first tries to match the input problem to some learned pattern from the first training step and if no pattern matches, the SVM prediction is used.
The author collected 500 problems, dividing them into 250 train and test samples and the combined approach achieved 75 % success rate on the training data. The SVM alone achieved 61 %.

The most obvious limitation of this solver is its reliance on defining the target equation template as a classification class. The solver can't predict any equation template which isn't predefined during training. While it's trivial to expand the model to new classes of problems, it's unclear how well it will perform, especially given the fact there can be a lot of new classes with a small number of training examples.

2.4 Datasets

This section describes the datasets used in the development and benchmarking of the current state-of-the-art solvers. The section is divided into three sections, based on the language of the data.

2.4.1 English data

Following the recent development in the field, larger benchmark datasets used to evaluate the current state-of-the-art solvers were collected only recently. Previously, the authors of the models tended to collect their own data for training and evaluation [ZWZ+19].

The oldest dataset still sometimes used to evaluate the state-of-the-art solvers is MAWPS(MAth Word ProblemS)[KKRA+16]. The dataset was published in 2016 and features problems from several smaller datasets. It contains problems of varying complexities, including multi-step problems and problems with irrelevant quantities. It allows choosing subsets based on minimal lexical similarity and overlap in equations. The lexical similarity between two problems is defined as the proportion of the intersection and union of their unigrams and bigrams. It features 3913 problems. Its arithmetic subset counts 2373 ones. Only this arithmetic subset is relevant for our work.

A newer frequently used English dataset is ASDiv(Academia Sinica Di- verse MWP Dataset) [MLS21]. Released in 2020, the corpus is drawn with the idea of maximizing lexicon usage. The supposed purpose is to make the dataset more challenging by restraining the ability to solve it using just mechanical/statistical pattern matching. The dataset contains 2305 MWPs of different types. The arithmetic subset ASDiv-a counts 1218 problems.
The data is annotated with problem level difficulties indicated by school level grades and type of the problem. The problem type is defined as the crucial thought pattern required to solve the problem. For arithmetic problems, the types are addition, subtraction, common-division, floor-division, ceil-division, sum, surplus, number-operation, and three different "Time-Variant Quantities"

- which mean that the solver should follow and sequentially update an entity-state variable.
The lexical diversity for a problem P in the context of the whole dataset D is defined in [MLS21] as:
1 - the largest BLEU score between P and the rest of the problems in D.
Experiments confirmed the higher difficulty of the new dataset, pointing out that higher lexical and problem type variation provides an additional challenge to solvers.

Despite ASDiv's attempt to constrain shallow pattern matching, according to experiments described previously in the chapter on shallow heuristics, it is still possible to achieve good results without really "understanding" the text, even though not as good as on MAWPS.

Based on these experiments and what authors of [PBG21] see as necessary qualities any MWP solver should possess, the authors create a new challenge dataset called SVAMP (Simple Variations on Arithmetic Math word Problems).
SVAMP is created by taking one hundred seed examples from ASdiv-a and applying simple variations on the problem texts so that it should be harder to exploit superficial patterns without making the problem more difficult for humans.

CATEGORY	VARIATION	EXAMPLES
Question Sensitivity	Same Object, Different Structure	**Original:** Allan brought two balloons and Jake brought four balloons to the park. How many balloons did Allan and Jake have in the park? **Variation:** Allan brought two balloons and Jake brought four balloons to the park. How many more balloons did Jake have than Allan in the park?
	Different Object, Same Structure	**Original:** In a school, there are 542 girls and 387 boys. 290 more boys joined the school. How many pupils are in the school? **Variation:** In a school, there are 542 girls and 387 boys. 290 more boys joined the school. How many boys are in the school?
	Different Object, Different Structure	**Original:** He then went to see the oranges being harvested. He found out that they harvest 83 sacks per day and that each sack contains 12 oranges. How many sacks of oranges will they have after 6 days of harvest? **Variation:** He then went to see the oranges being harvested. He found out that they harvest 83 sacks per day and that each sack contains 12 oranges. How many oranges do they harvest per day?
Reasoning Ability	Add relevant information	**Original:** Every day, Ryan spends 4 hours on learning English and 3 hours on learning Chinese. How many hours does he spend on learning English and Chinese in all? **Variation:** Every day, Ryan spends 4 hours on learning English and 3 hours on learning Chinese. If he learns for 3 days, how many hours does he spend on learning English and Chinese in all?
	Change Information	**Original:** Jack had 142 pencils. Jack gave 31 pencils to Dorothy. How many pencils does Jack have now? **Variation:** Dorothy had 142 pencils. Jack gave 31 pencils to Dorothy. How many pencils does Dorothy have now?
	Invert Operation	**Original:** He also made some juice from fresh oranges. If he used 2 oranges per glass of juice and he made 6 glasses of juice, how many oranges did he use? **Variation:** He also made some juice from fresh oranges. If he used 2 oranges per glass of juice and he used up 12 oranges, how many glasses of juice did he make?
Structural Invariance	Change order of objects	**Original:** John has 8 marbles and 3 stones. How many more marbles than stones does he have? **Variation:** John has 3 stones and 8 marbles. How many more marbles than stones does he have?
	Change order of phrases	**Original:** Matthew had 27 crackers. If Matthew gave equal numbers of crackers to his 9 friends, how many crackers did each person eat? **Variation:** Matthew gave equal numbers of crackers to his 9 friends. If Matthew had a total of 27 crackers initially, how many crackers did each person eat?
	Add irrelevant information	**Original:** Jack had 142 pencils. Jack gave 31 pencils to Dorothy. How many pencils does Jack have now? **Variation:** Jack had 142 pencils. Dorothy had 50 pencils. Jack gave 31 pencils to Dorothy. How many pencils does Jack have now?

Figure 2.9: Variations applied in mutating the ASDiv seed examples in the creation of SVAMP, reprinted from [PBG21].

2.4.2 Chinese data

As large bulk of current research is done by researchers from China, the largest and most used datasets are in Chinese.

Along with MAWPS, the typical benchmark dataset used since the usage of deep learning went mainstream is Math23k. It was collected for the training of the pioneering deep learning seq2seq solver DNS and it features 23162 arithmetic problems collected from different online education websites. The knowledge of the value of pi by the solver is assumed. Additionally, unlike in the English datasets, it is assumed that apart from the quantities in the problem, the solver can also use the constants 1 and pi. The featured paper contains no information on the lexical and problem type diversity of the data.

Newer dataset Ape210k presented in 2021 in [LZSZ21] features 130000 us-

able problems, some requiring additional common-sense background knowledge. However, according to [ape], the dataset is not publicly available. It is still important to mention as [LZSZ21] achieves an major improvement also in Math23k when trained on it.

2.4.3 Czech data

The creators of [KP20] collected 500 AWPs from Czech textbooks. Their difficulty is up to third grade and each of them is solvable in at most two steps. The problems can contain one irrelevant quantity. There are ten types of possible equation templates, dividing the dataset into problem classes.

2.5 Comparison

Now that we have presented both the various approaches used to solve MWPs and the currently used data, it's time to compare how successful the approaches are on current data.

To appreciate the rapid development in recent years, we present results from state-of-the-art survey from the year 2019. AI2, CC and IL are smaller datasets with problems similar to those in MAWPS, which used many of their problems. The Metric used is whether the produced equation template evaluates to the correct numeric value. The results are in percentages.

			AI2	IL	CC	SingleEQ	AllArith	Dolphin-S	MAWPS-S	Math23K
# problems			395	562	600	508	831	7,070	2,373	23,162
operators O			$\{+,-\}$	$\{+,-,\times,\div\}$	$\{+,-,\times,\div\}$	$\{+,-,\times,\div\}$	$\{+,-,\times,\div\}$	$\{+,-,\times,\div\}$	$\{+,-,\times,\div\}$	$\{+,-,\times,\div\}$
Statistic-based	ARIS [25]	2014	77.7	-	-	48	-	-	-	-
	Schema [26]	2015	**88.64**	-	-	-	-	-	-	-
	Formula [27]	2016	86.07	-	-	-	-	-	-	-
	LogicForm [28], [29]	2016	84.8	**80.1**	53.5	-	-	-	-	-
Tree-based	ALGES [31]	2015	52.4	72.9	65	72	60.4	-	-	-
	ExpressionTree [30]	2015	72	73.9	45.2	66.38	79.4	26.11	-	-
	UNITDEP [32]	2017	56.2	71.0	53.5	**72.25**	**81.7**	28.78	-	-
DL-based	MathDQN [17]	2018	78.5	73.3	**75.5**	52.96	72.68	30.06	60.25	-
	Seq2SeqET [15]	2018	-	-	-	-	-	-	-	66.7
	T-RNN [19]	2019	-	-	-	-	-	39.1	66.8	66.9
	StackDecoder [16]	2019	-	-	-	-	-	-	-	65.8

Figure 2.10: State-of-the-art from 2019 survey, reprinted from [ZWZ+19].

Next we present the results of the state-of-the-art solvers presented in the chapter on the third phase of development.

	MAWPS	ASDiv-a	SVAMP	Math23k
GTS	82.6	71.4	30.8	74.3
GTS-R	88.5	81.2	41.0	
Graph2Tree	83.7	77.4	36.5	75.5
Graph2Tree-R	88.7	82.2	43.8	
Generate-Rank	84.0			84.3
MWP-Bert				82.0
MWP-Bert Ape				96.2
Gri	93.7			

Table 2.3: Comparison of state-of-the-art models.

GTS-R and Graph2Tree-R are the same models with pre-trained RoBERTa embeddings as inputs, as used in [PBG21]. RoBERTa [LOG+19] is a BERT-like large pre-trained language model.
"Mwp-bert Ape" refers to the Mwp-bert solver trained on both Math23k and Ape210k. However, due to Ape210k not being public, it's not possible to see how well it deals with other datasets, most notably SVAMP.
Generate-Rank's result on Math23k illustrates its supposed improved ability in solving problems with longer equations, as those are more present in Math23k. The performance of GTS-R, Graph2Tree-R, and MWP-Bert provides evidence for the power of pre-trained language models.

Chapter 3

Method

In this chapter we discuss our approach to adapt the state-of-the-art methods of MWP solving into the Czech language. We describe our data collection and processing, used solvers and planned experiments.

3.1 Used Data

In terms of native Czech data, we have only five hundred MWPs from the WP500CZ dataset. We assume this is not enough to train the data demanding deep-learning based solvers.

To overcome this limitation, we turn to machine translation. The rapid expansion of deep-learning based natural language processing provided us with decent quality automatic translations, which may be suitable for our task. It may also be interesting to check how the results are affected by different translators. Trying multiple translators may not only give us the advantage of choosing the one best suited for our task, but it also may be an interesting metric for the quality of the translator itself. This is especially interesting considering that the largest datasets are in Chinese, and translating from Chinese still seems to be a difficult task.

In our experiments, we will use the Czech translations obtained using the DeepL translator [1] of the mentioned English datasets: the arithmetic section of MAWPS (we will address this section just as MAWPS in the rest of our work), ASDiv-a and SVAMP. We will also use their original versions to compare the performances.

We will use the native Czech dataset WP500CZ for both training our models and native data evaluation.

Additionally, we will translate the Chinese Math23k dataset into English and if we find the results satisfactory, we may further attempt to leverage its Czech translation.

3.2 Translation

For English to Czech translation, we decided to use the DeepL translator, which is generally considered to provide slightly better translations than the best known Google translator [col22][tea21]. The translated texts were easily understandable and generally without any unnaturally sounding formulations. For this reason, we decided to not use other translators.
The translator sometimes changed the formatting of the numbers. The most notable one is reformating numbers with four or more digits by adding whitespace after each multiply of thousand. For example, ”12345” was turned into ”12 345”. This introduced ambiguity, as it’s not clear whether the string refers to one larger number or more smaller ones. Since the solvers are evaluated on data with already tagged quantities, we resolved this ambiguity manually. The translation also rarely produced other miscellaneous quirks, one example is replacing hours in English time format (0-12 am, 1-12 pm) with Czech one (0-24). This can invalidate the problem if the hour numbers are present in the equation.

[1] https://www.deepl.com/, translation of English datasets done in August 2021, the English to Czech for Math23k in May 2022.

English text	”Having been to Paris, Rob also remembered the Eiffel Tower which was the tallest structure in the world at the time it was built. If the height of Eiffel Tower is 324 m, how much lower is it compared to 830 m height, today’ s tallest man-made structure, the Burj Khalifa?”
Czech text	”Po návštěvě Paříže si Rob vzpomněl také na Eiffelovu věž, která byla v době svého vzniku nejvyšší stavbou na světě. Pokud je výška Eiffelovy věže 324 m, o kolik je nižší ve srovnání s 830 m vysokou, dnes nejvyšší stavbou vytvořenou člověkem, Burdž Chalífou?”

Table 3.1: Example of MWP translated automatically from English to Czech.

We found that the quality of Chinese to English automatic translation is significantly worse. We tried several translators and decided to use Baidu [2], which according to our opinion produced the most understandable results. We also tried the NLP-Helsinki Opus translation system [TT20], to compare how the quality of translation influences the solver performance.

Chinese text	空调厂准备装配一批空调，计划每天装配 45 台，20 天完成. 实际 18 天就完成了任务，实际每天装配多少台?
English text, Baidu	”The air conditioning factory is preparing to assemble a batch of air conditioners. It plans to assemble 45 sets every day and complete them in 20 days. In fact, the task is completed in 18 days. How many sets are actually assembled every day?”
English text, Helsinki	”The air-conditioning plant prepares to assemble a group of air-conditioning units, and plans to assemble 45 units per day, 20 days to complete. The actual 18 days have been completed, and how many units per day have been assembled?”

Table 3.2: Example of MWP translated automatically from Chinese to English.

3.3 Data processing

The data in their raw form are not suitable for direct use. At the bare minimum, we need to detect words expressing quantities and convert them into their

[2]https://fanyi.baidu.com/, used through Python API during autumn 2021.

respective values. Other complications depend on the specifics of the dataset, and we found some datasets contain problems that aren't usable in our work at all.

To our awareness, all of the state-of-the-art solvers have their own data preprocessing procedures. Some authors [GK21] make their preprocessing procedure public along with the model code, others [XS19][ZWL+20][PBG21] include either partially or fully processed data themselves but not the whole procedures. In either case, neither is directly usable for the translated data.
The primary goals of the data preprocessing are:

- Convert numeric words into their numeric quantities. We call this task *number tagging* and the procedure devised the *number tagger*.
- Detect unusable problems and either correct them or discard them.
- In the case of MAWPS convert equations into usable explicit forms.

We based our approach on the number tagging on English data from [GK21] and general text preprocessing from [GK21] and [XS19]. We expanded them to work on Czech data and to correct or filter unusable problems. Criteria for considering a problem unusable are specified later.

3.3.1 Number Tagging

The task of detecting natural language words expressing quantities and translating them into their numeric values isn't completely trivial. [GK21] uses the Python library word2number [w2n], which is based on dictionary matching of exact words. Methods used by [XS19][ZWL+20][PBG21] are not disclosed, they provide the data with this step already done.

The dictionary matching approach has several drawbacks. The number of rules needed to separate all words expressing quantities without an error would be enormous if that is even possible. For example, consider the Czech word "sedmikráska" (daisy, literally something like "sevenbeauty"). Other difficulties are

misspelled words, merging with punctuation and similar ambiguities. Also, tagging every number may be counter-productive to the solver's performance, as it may tag quantities that are almost surely irrelevant to the solution.

Employing a context-aware number tagger may provide a little benefit to the final performance and generalisability to new data and on a more general level, perhaps it should be considered an ability the solver should possess itself. But we believe that for the aims of this thesis, the simple dictionary matching procedure is sufficient.

The word2number library doesn't support the Czech language, however, it provides means for a relatively simple extension. We developed it for our number tagging in the Czech language. Additionally, we included several new rules to deal with frequent omissions in both Czech and English. For example, the original word2number does not translate "twice", a word that appeared often in the data. We added a rule to translate it into "2 times".

3.3.2 Problem Selection

In this subsection we discuss criteria for discarding a problem separately for each dataset. The basic necessities for a problem included in our final data are:

- The equation can't contain numbers which the number tagging procedure haven't found in the problem text.
- The expression evaluates into the provided answer.
- The problem isn't a duplicate of another problem in its dataset.

MAWPS

Out of the English datasets, processing the MAWPS is the least trivial and our selection ended up being substantially different from the ones authors of

the solvers we use extracted.

Significant amount of problems are dropped from MAWPS due to introduction of numbers in the equation which weren't tagged in the problem text. This includes:

- Problems requiring background knowledge, most often working with percentages or converting cents to dollars.
- Number tagger failures, for example the word half is connected to 0.5 in the equation but isn't tagged in the text or the word dozen isn't converted to 12. Dozen is also wrongly translated few times.
- The number in equation is inverse of the number in the problem text. For example, for the problem: "The sum of a number and its reciprocal is 3.3333. Find the number.", the provided equation is "number + (1.0 / number) = 3.3333"
- A specific type of MWP we discovered, which to our knowledge weren't addressed in any of our source texts. It's a type where the quantity isn't explicitly mentioned, but it's expected that the solver will derive it from the logic of the problem. For example, consider the following MWP:

  "After eating at the restaurant, Sally, Sam, and Alyssa decided to divide the bill evenly. If each person paid 45 dollars, what was the total of the bill ?"

  The correct equation is $x = 45 * 3$, but the quantity '3' has to be deduced from the fact that the text presents three people paying the bill.

  We call these problems *problems with implicit quantities* and the current solvers can't robustly solve them, because they require the ability to output arbitrary constants. We believe that their robust solving presents an interesting research direction, as we would consider making these deductions an elementary numeracy skill.

MAWPS also presents a substantial amount of problems with an equation in which the queried variable isn't expressed explicitly. We try to salvage as many

Body and Question	The Hudson River flows at a rate of 3 miles per hour. A patrol boat travels 60 miles upriver , and returns in a total time of 9 hours. What is the speed of the boat in still water?
Expression	(200.0 / speed) - 1.0 = 200.0 / (10.0 + speed)

Table 3.3: Example of a problem without explicitly expressed variable. The explicit expression found by our procedure contained square root, therefore was unusable and the problem was discarded.

of these problems using symbolic manipulation with the Python library sympy [MSP+17]. Despite our best efforts, we weren't always successful. Some of the equations can't be expressed without adding the constant '1' due to factoring out the queried variable. Another introduce square root into the expression. Some of the equations can also end up being quite complicated relative to the rest of the problems. We decided to keep them anyway. We also found significant amount of problems with wrong solution or equation. We corrected those manually and included them in the final set.

To summarize, of the 2373 problems in the Czech translation of the dataset:

- 176 were dropped due to having numbers in their equations which weren't tagged in the problem text.
- 26 were dropped due to failure in converting their equation into explicit form without new symbols.
- 167 were duplicates of other problems.

For the final Czech part, 1996 problems were retained.
For the English part, the number before dropping duplicates was marginally higher, however, the number of duplicated problems was 227, leading to only 1947 problems in the final set. This difference is caused by the fact that translations of the same text weren't always equal, possibly due to the introduction of some randomness in the translating engine.
Compared to the selection present in [PBG21], our English part retrieved 172 more problems, excluding the duplicates in the [PBG21] selection.

ASDiv-a

From ASDiv-a, the arithmetic subset of ASdiv, we decided to drop the floor-division and the ceil-division problems, as their correct answers are computed as integer division, which is a different operation from the standard division used in the rest of this work, practically amounting to a new operator.
Other examples were dropped because of background knowledge expectations, the dataset contains several problems typically requiring the knowledge of the number of some specific body part some creature has, ie. a dog has four legs. There are also several problems requiring implicit deduction, as specified before. Summarily, out of the 1217 problems, we retained 1179 Czech problems and 1178 English problems.

Math23k

This Chinese dataset contains 23163 problems, making it by far the largest one we work with. The original Chinese data doesn't need number tagging, as all the relevant numbers are already converted. The problem texts provide even percentages and inverse numbers in clear form. This however isn't true for the translated versions. Large number of problems require adding the constant one or 3.14 (pi). We decided to drop those problems due to consistency with the rest of our data, where introducing untagged numbers in the equation is forbidden.
These problems along with others dropped for untagged equation numbers compose most of the dropped problems. Due to large amount of dropped problems and generally not so great quality of the translations, we haven't analyzed reasons for the untagged numbers them further, but generally we expect main causes come from the translation.

In summary, we recovered 15315 problems from the set translated by Baidu. From the Helsinki set we recovered 13898.

We decided to translate further into Czech only the Baidu set. Additional 316 problems from the Czech translation were dropped, mostly due to translation or number tagger errors. Therefore our Czech subset counts 14999 problems.

Others

Preprocessing the remaining two datasets proved to be more straightforward. WP500CZ contained three problems with a wrong solution and three problems with a wrong equation. We trivially corrected them. Therefore we retained all 500 native Czech MWPs.

The SVAMP dataset had three problems with wrong solutions. Two were corrected and one was discarded due to us not being quite sure what is the correct answer.

Duplicate detection is also necessary when combining datasets. We found that our selection of MAWPS and ASDiv-a share 99 MWPs in Czech sets and 188 problems in English sets. The difference is again due to the small differences in translations of the same problems.

Finally, it is worth noting that we encountered a few additional miscellaneous ambiguities during the text parsing, which we tried to resolve manually. For example, a number with a minus in the equation can be present as a negative number in the text.

At the end of the section dedicated to our data, we present a summary table including some statistics.

Dataset	n problems	n problems after preprocessing	duplicates	avg tagged numbers	avg equation size
MAWPS	2373	1947	0	2.70	4.04
MAWPS CZ	2373	1996	0	2.64	4.00
MAWPS SH	2373	1921	149	2.5	3.90
MAWPS Gri	2373	1901	83	2.54	3.82
ASDiv-a	1218	1178	0	2.32	3.40
ASDiv-a CZ	1218	1179	0	2.31	3.40
ASDiv-a SH	1218	1217	0	2.36	3.46
SVAMP	1000	999	0	2.83	3.47
SVAMP CZ	1000	999	0	2.83	3.47
WP500CZ	500	500	0	2.13	3.30
Math23k-Zh	23162	15315	0	2.94	5.25
Math23k-B	23162	15315	0	3.19	5.28
Math23k-H	23162	13898	0	3.25	5.18
Math23k-CZ	23162	14999	0	3.16	5.22

Table 3.4: Summary of used dataset selections. SH presents selections provided in the paper on shallow heuristics [PBG21]. Gri presents selections used by [GK21]. Math23k-Zh is our selection of original Chinese problems. We kept only those for which we recovered their translation in the Baidu set. Math23k-B is the set translated by Baidu, -H by Helsinki.

3.4 Solvers

We have chosen two baseline solvers to run our experiments on. First, we will use the [GK21] transformer approach, since it's simple and fast to train, therefore giving us enough options to experiment with different versions of datasets and translations. According to the authors, the model provides even better performance on the MAWPS dataset than the more complex ones, however, it was tested only MAWPS and a few older smaller datasets, leaving its performance on the more challenging data unexplored. Considering the relative simplicity of the approach, we don't expect its primacy to hold.

The approach consists of using a vanilla transformer model to translate the problem text into an equation in postfix form. They also experimented with several new preprocessing steps, from which a procedure called "label-selective

tagging" proved to be the most successful. The procedure is meant to reduce the amount of tagged irrelevant quantities by searching for the most common terms in the problem text and if the problem question contains them, it tags only numbers relevant to them. We will call this model "Gri" further in our text.

The second solver we will try is the GTS. We chose this one instead of the similar and marginally superior Graph2Tree because it doesn't require the complex preprocessing steps required to transform the problem text into its graph representation. We also believe its difference with Mwp-bert is insignificant when pre-trained language models are used to provide the input embeddings, as done in [PBG21].

The best results seem to be currently achieved by combining a large pre-trained, transformer-based language model providing encoded embeddings of the input text and a specialized method for decoding, as seen with Mwp-bert and experiments in [PBG21]. In the third phase, we will try to replicate these results using the approach from [PBG21].
This replication requires the usage of language models trained on the Czech language. We test two multilingual versions of classic models, multilingual BERT (mBERT) [DCLT18] and XLM-RoBERTa [CKG+19] and a BERT-like model trained specifically on Czech, Czert [SPP+21].

3.5 Experiment design

The primary goal of our experiments is to train a solver capable of achieving the best performance on the native Czech dataset WP500CZ, which we consider as proof that the state-of-the-art solvers can be well adapted to native Czech even with limiting amount of native data. Secondarily, if we observe similar performance on both English and its Czech translation, this may give us an interesting hint on the quality of the translation itself. Especially if we also find that the solvers trained only on the translated problems work well on the native data too. This is especially interesting for translations from Chinese, which are both more challenging and more important in our contemporary reality.
In the first phase, which we call *data analyzing phase*, we will:

- Compare the results of the baseline models Gri, GTS on the MAWPS, ASDiv-a and SVAMP selections provided in [PBG21]. We denote the authors and their selections with SH (shallow heuristics). We will do the same with the MAWPS selection used in the Gri paper. The goal is to find how the differences in preprocessing affect performance and also to evaluate Gri on the more challenging data.
- Compare the performance achieved on our English selections of MAWPS, ASDiv-a and SVAMP with their machine translated Czech counterparts. We hope the performances will be similar to each other, which would provide us a strong hint that the translations retain their logical structure.
- We will also try to use the two English translations of the Math23K dataset, analogously first comparing the performance difference between Chinese and English versions, and if the result is satisfying, further translate into Czech.

In the second phase, which we call *adapt to native Czech phase*:

- We will train the baseline models on the native Czech dataset WP500CZ alone, to receive a baseline upon which we can build further.
- Then we will test the solvers trained only on the translated data on the native data, hoping they can generalize well.
- Finally, we will train on both the data translated into Czech and the train split of the Czech native data and hope this joint training helps us improve the baseline.

And in the final phase, called *unleash the power of pre-trained language models phase* we will try to replicate the results from [PBG21] on the translated data. First, we will use BERT and RoBERTa on the English data. Then we will try the multilingual models XML-RoBERTa and BERT on both English and translated and finally try the Czech BERT model Czert on the Czech data, hoping we can achieve improvements analogous to those of SH.

Chapter 4

Experiments

4.1 Implementation details

We used the default hyperparameters provided by their authors for both models. For Gri, our work expanded on the publicly provided code. For GTS, we based our experiments on the setup code provided by SH.

Authors of Gri found that their model performed best using only two transformer layers, $d_{model} = 256$, eight attention heads and 0.1 dropout. They trained the model using sparse cross-entropy loss and used the Adam optimizer with beta $\beta_1 = 0.95$ and $\beta_2 = 0.99$ with a standard epsilon of 1e−9 with batch size of 128 problems. We trained the models over 200 epochs without further experiments with this setup.

For GTS, we expand on the public implementation provided by SH. The model is trained using cross-entropy loss over predicted and target sequence. The used optimizer is Adam. The encoder is a 2-layer bidirectional LSTM. Due to relatively high computational demand, we decided to limit the training of the models only to 20 epochs, which we found sufficient for achieving results analogous to ones found by SH. The input embedding module is either trained from scratch using the PyTorch Embedding module or provided by a pre-trained

	scratch	pre-trained
batch size	16	4
embedding size	128	depends on the model
hidden size	512	512
learning rate	2e-3	1e-2
embedding learning rate	2e-3	8e-6
dropout	0.5	0.5

Table 4.1: Hyperparameters used for the GTS.

model.
We use the hyperparameters tested by the same authors, which are different for the variant trained with embedding provided by the pre-trained model and the one trained from scratch.

The pre-trained models were provided by the HuggingFace Transformers library [WDS+20]. Unless specified otherwise, all results were obtained using five-fold cross-validation or three-fold in the case of experiments featuring Math23k. All results are provided as percentages. The metric used is whether the produced equation is evaluated into the correct numeric value.

4.2 Results

4.2.1 Data analyzing phase

The first part is comparing performance of the baseline models with our English data selections with the selections used by authors of Gri and SH.

	MAWPS Gri	MAWPS SH	MAWPS	ASDiv-a SH	ASDiv-a	SVAMP SH	SVAMP
GTS	91.4	84.3	81.8	68.5	72.8	30.8*	32.5
Gri	93.2	73.4	69.2	42.8	45.5	15.9	17.1

Table 4.2: Performance of baseline models on different data selections. Performance on Gri is on the train-test split provided by authors instead of cross-validation. The numbers represent percentages of predicted equations that are evaluated into the correct numeric value. * Is taken from [PBG21]

The immediately noticeable thing is the suspiciously high performance on the selection provided by Gri. That result is however consistent with their claim.
When we analyzed the data, we found that they select the first 200 problems for the test set without shuffling. This is probably the source of the bias responsible for the high score, as the problems from the beginning are simpler, for example, there isn't a single problem requiring multiplication or division.
Sadly, this explains why this relatively simple model was able to achieve the state-of-the-art result on MAWPS and the worse performance on the remaining data.

Another notable thing is that while we expected the models to fare worse on our selection of the MAWPS, we expected the difference to be larger.

Now we present the comparison between the performance on our English and Czech selections.

	MAWPS	MAWPS CZ	ASDiv-a	ASdiv-a CZ
GTS	81.8	80.4	72.8	70.2
Gri	69.2	60.8	45.5	35.6

Table 4.3: Table comparing the results in % of our solvers on the data in English and their translated Czech equivalents.

We see that in the case of GTS, the performance is similar. On Gri, it seems to degrade more significantly. As of now, we don't have a good hypothesis why this happened.
And finally, we conclude the first phase by evaluating the translations from Chi-

nese. Since training on relatively large amounts of problems is computationally demanding, we do only three-fold cross-validation and test only on the GTS.

Math23k version	Score (%)
Chinese	67.7
English (Baidu)	59.5
English (Helsinki)	51.7
Czech	56.8

Table 4.4: Results for Math23k. Tested on our selection of the original Chinese problems, the two versions of Chinese to English translation and the English (Baidu) translation to Czech translation.

The result for the Baidu translation show more substantial drop compared to native English-Czech translations. However, we consider the result to be good enough to warrant further work with the data.

We interpret the worse results on the set translated by Helsinki to be due to the worse quality of the translation. We decide to not use it further.

4.2.2 Adapt to native Czech phase

Our experiment with training on the WP500CZ turned very different for both models:

GTS achieved 71.8% accuracy, while Gri only 24.4%. This could further support the hypothesis that the model has problem with Czech data. However since it was trained from scratch, it's not clear why that would be the case. Alternatively, the fact that WP500CZ, with its 500 examples, is the smallest dataset in our work could have effect too, though we find it hard to believe the impact would be this large.
Neverless, the good accuracy of the GTS on such small amount of data is perhaps a bit surprising too.

In the next table, we present the generalization experiments, the results of solvers trained on translated Czech data, evaluated on the whole native WP500CZ

dataset. We proceed to use the GTS solver only.

Dataset	Score (%)
MAWPS CZ	56.3
ASDiv-a CZ	56.2
SVAMP CZ	37.6
MAWPS CZ + ASdiv-a CZ	61.2
MAWPS CZ + ASdiv-a CZ + SVAMP CZ	61.7
Math23k CZ	28.5
All CZ	46.7

Table 4.5: Generalization experiments - results on the whole WP500CZ for solver trained on the translated data.

. The results are decent enough to prove that the solvers trained on the translated problems do indeed generalize on the native data. In our opinion, this is an interesting result, as it could point out multiple things - that the models really exploit similar patterns in both native English and native Czech MWPs and the translation preserves them? Alternatively, the translation is capable of creating the same patterns inherent in native Czech data?
The worse generalization of the solver trained on SVAMP, the dataset designed to eliminate shallow patterns, suggests the former.

We found the generalization of the Math23k Czech translations was quite disappointing. We haven't expected them to generalize well, given both the often spurious translations and contrasting statistics of the data (the set with longest average vs. shortest average equation length), but the result is still a little surprising. We decided to not work further with them.

And finally, we present the evaluation on solver trained on MAWPS CZ, ASDiv-a CZ, SVAMP CZ and 350 problems from the native WP500CZ, tested on the remaining 150 problems in WP500CZ.

We achieved an accuracy of 79.8%, which constitutes better result than the 75% from [KP20]. However, the results are not directly comparable, because [KP20] used different train and test sets. It is also an eight percent improvement from training on WP500CZ alone.

4.2.3 Unleash the power of pre-trained language models phase

In the last phase, instead of training the input word embeddings of the GTS from scratch, we try to provide word embeddings taken from the large transformer-based pre-trained language models. We use BERT and RoBERTa on the English data for the baseline and their multilingual versions mBERT and XML-RoBERTa on all our data sets. We also try the BERT like model Czert, trained only on Czech. The results are presented in the following tables.

	MAWPS	MAWPS CZ	ASDiv-a	ASDiv-a CZ	SVAMP	SVAMP CZ	WP500CZ
baseline	81.8	80.4	72.8	70.2	33.0	32.2	71.8
BERT	84.3	-	77.9	-	34.6	-	-
RoBERTa	85.5	-	79.4	-	52.2	-	-
mBERT	84.1	80.1	77.3	67.4	35.1	27.6	69.0
XLM-RoBERTa	x	77.3	64.1	x	25.7	20.0	x
Czert	-	81.1	-	68.1	-	29.8	65.4

Table 4.6: Comparison of performance of solvers with input embeddings provided by pre-trained models. All results are provided as percentages. The metric used is whether the produced equation is evaluated into the correct numeric value. 'x' means we chose to not run the experiment.

Results for the English data selections show the beneficiality of this approach, however we failed to unleash their power and demonstrate any improvement on the Czech data. It is worth noting that the multilingual model mBERT were tried on both Czech and English data with the same setup, providing benefit only for solving the English problems. Also the performance of solver with XLM-RoBERTa showed to be underwhelming. We could try experimenting with different hyperparameter settings or more training epochs, however we decided to pursue more promising directions instead and dropped the remaining evaluations.

In the next step we measure how the pre-trained embeddings influence results of the generalization experiments.

Data	baseline	mBERT	Czert
MAWPS	56.3	57.0	60.0
ASDiv-a	56.2	58.0	58.9
MAWPS + ASDiv-a	61.2	65.1	67.6
SVAMP	37.6	41.7	43.6
MAWPS + ASDiv-a + SVAMP	61.7	63.8	66.7

Table 4.7: Results in % of the solvers trained on the translated data and tested on the whole native WP500CZ dataset.

While the employment of pre-trained models provided no performance benefit during the evaluation on the translated data alone, we see that they provide significant benefit on generalization.
We hypothesize this is due to tokenization. The variant trained from scratch tokenizes all words it hasn't encountered during training as the <unk> unknown token, while we expect that the pre-trained models have vocabularies spanning space well beyond what was seen during training.

And finally, we repeated our last experiment with solver trained on MAWPS CZ, ASDiv-a CZ, SVAMP CZ and 350 problems from the native WP500CZ, evaluating the solvers on the remaining 150 problems in WP500CZ.

Solver	Score (%)
baseline	79.8
Czert	78.0
mBERT	83.5

Table 4.8: Generalization experiments, including - results on the whole WP500CZ for solver trained on the translated data.

. We believe these results provide more evidence that our approach outperforms [KP20].
However, we believe our work makes two more significant contributions than a few percentage points in this benchmark.
Firstly, we provide evidence that turning to machine translation is a valid direction in case of scarcity of high-quality data. We document this by the decent generalizations of solvers trained on the translated data, and by the improvement we see using the translated data along with native ones.

Secondly, we provide an approach that is much simpler to generalize to more difficult MWPs than the ones featured in the native Czech dataset. Our solvers can potentially solve MWPs without any limitation on the equation template and are already trained on a more diverse set of MWPs. We believe this also means our approach has more perspective to facilitate further developments in applications requiring more general forms of numeric reasoning, for example in the development of automatic fact-checkers.

Chapter 5

Conclusion

The goal of our work was to evaluate the performance of state-of-the-art math word problem (MWP) solvers on the Czech language. We studied the history of the field and found out that its future lies in the employment of the modern deep-learning-based methods. The currently most promising direction seems to be in combining the transformer-based pre-trained language models with domain-specific architectures.

We found only one MWP dataset in native Czech, counting 500 MWPs. We considered this amount insufficient for full utilization of the power of deep learning. To address this limitation, we turned to machine translation. The DeepL translator provided a clearly understandable translation from English to Czech. However, the largest amount of data is in Chinese. Despite our best attempts to choose the best translator, the translations obtained by our choice, Baidu, proved to be dubious sometimes.

We experimented with a relatively simple transformer model and a more complex solver GTS. Performance of the simple model dropped significantly on the translated data, however, for GTS, the English to Czech translation proved to have only little effect. The drop in Chinese to English translation was more notable.

Using the GTS we achieved decent results on the native Czech dataset. The GTS trained only on the native data achieved surprisingly high accuracy, 71.8%. Decent accuracy, over 60%, was also achieved with solvers trained only on the data translated from English, proving the translations generalize. Combined

training on both native and translated problems achieved results up to around 80%.

Our experiments with the pre-trained models proved their power in English, however, lead to no significant improvement in Czech when evaluated on the same dataset they were trained. However, we found that when trained on the translated data and evaluated on the native, they perform slightly better.

We believe the most unique contribution of our work lies in demonstrating that machine translation can be a valid alternative for researchers lacking a sufficient quantity of data. We hope our work could provide a useful resource for researchers wishing to explore various numeracy skills of their machine learning models further.

Appendix A

Bibliography

[Ala] Jay Alammar, *The illustrated transformer*, `https://jalammar.github.io/illustrated-transformer`, Accessed: 2022-05-18.

[ape] *Ape210k: A large-scale and template-rich dataset of math word problems*, `https://ui.adsabs.harvard.edu/abs/2020arXiv200911506Z/abstract`, Accessed: 2022-05-20.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, *Neural machine translation by jointly learning to align and translate*, arXiv preprint arXiv:1409.0473 (2014).

[BGMMS21] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell, *On the dangers of stochastic parrots: Can language models be too big?* , Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, 2021, pp. 610–623.

[CKG+19] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov, *Unsupervised cross-lingual representation learning at scale*, CoRR **abs/1911.02116** (2019).

[col22] *Deepl vs google translate: Which is better? + how to use them (2022)*, `https://translatepress.com/`

deepl-vs-google-translate-comparison/, Feb 2022, Accessed: 2022-05-18.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, arXiv preprint arXiv:1810.04805 (2018).

[GK21] Kaden Griffith and Jugal Kalita, *Solving arithmetic word problems with transformers and preprocessing of problem text*, arXiv preprint arXiv:2106.00893 (2021).

[Hoc98] Sepp Hochreiter, *The vanishing gradient problem during learning recurrent neural nets and problem solutions*, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **6** (1998), no. 02, 107–116.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber, *Long short-term memory*, Neural computation **9** (1997), no. 8, 1735–1780.

[KKRA+16] Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi, *Mawps: A math word problem repository*, Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2016, pp. 1152–1157.

[KP20] Jan Kadlec and Daniel Prusa, *Solvers for mathematical word problems in czech.*, ITAT, 2020, pp. 18–25.

[LOG+19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov, *Roberta: A robustly optimized bert pretraining approach*, arXiv preprint arXiv:1907.11692 (2019).

[LZSZ21] Zhenwen Liang, Jipeng Zhang, Jie Shao, and Xiangliang Zhang, *Mwp-bert: A strong baseline for math word problems*, arXiv preprint arXiv:2107.13435 (2021).

[MG08] Anirban Mukherjee and Utpal Garain, *A review of methods for automatic understanding of natural language mathematical problems*, Artificial Intelligence Review **29** (2008), no. 2, 93–122.

[MLS21] Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su, *A diverse corpus for evaluating and developing english math word problem solvers*, arXiv preprint arXiv:2106.15772 (2021).

[MSP+17] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz, *Sympy: symbolic computing in python*, PeerJ Computer Science (2017).

[PBG21] Arkil Patel, Satwik Bhattamishra, and Navin Goyal, *Are nlp models really able to solve simple math word problems?*, arXiv preprint arXiv:2103.07191 (2021).

[RVR15] Subhro Roy, Tim Vieira, and Dan Roth, *Reasoning about quantities in natural language*, Transactions of the Association for Computational Linguistics **3** (2015), 1–13.

[SK15] Sowmya S Sundaram and Deepak Khemani, *Natural language processing for solving simple word problems*, Proceedings of the 12th International Conference on Natural Language Processing, 2015, pp. 394–402.

[SPP+21] Jakub Sido, Ondřej Pražák, Pavel Přibáň, Jan Pašek, Michal Seják, and Miloslav Konopík, *Czert–czech bert-like model for language representation*, arXiv preprint arXiv:2103.13031 (2021).

[SYL+21] Jianhao Shen, Yichun Yin, Lin Li, Lifeng Shang, Xin Jiang, Ming Zhang, and Qun Liu, *Generate & rank: A multi-task framework for math word problems*, arXiv preprint arXiv:2109.03034 (2021).

[tea21] *Here is how deepl translator and google translate compare*, `https://revolutionized.com/deepl-translator-vs-google-translate/`, Oct 2021, Accessed: 2022-05-18.

[TPSI21] Avijit Thawani, Jay Pujara, Pedro A Szekely, and Filip Ilievski, *Representing numbers in nlp: a survey and a vision*, arXiv preprint arXiv:2103.13136 (2021).

[TT20] Jörg Tiedemann and Santhosh Thottingal, *OPUS-MT —Building open translation services for the World*, Proceedings of the 22nd Annual Conferenec of the European Association for Machine Translation (EAMT) (Lisbon, Portugal), 2020.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, *Attention is all you need*, Advances in neural information processing systems **30** (2017).

[w2n] *word2number*, `https://w2n.readthedocs.io/en/latest/`, Accessed: 2022-05-18.

[WDS+20] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush, *Transformers: State-of-the-art natural language processing*, Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (Online), Association for Computational Linguistics, October 2020, pp. 38–45.

[WLS17] Yan Wang, Xiaojiang Liu, and Shuming Shi, *Deep neural solver for math word problems*, Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, 2017, pp. 845–854.

[XS19] Zhipeng Xie and Shichao Sun, *A goal-driven tree-structured neural model for math word problems.*, IJCAI, 2019, pp. 5299–5305.

[ZDLS20] Ming Zhou, Nan Duan, Shujie Liu, and Heung-Yeung Shum, *Progress in neural nlp: modeling, learning, and reasoning*, Engineering **6** (2020), no. 3, 275–290.

[ZWL+20] Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim, *Graph-to-tree learning for solving math word problems*, Association for Computational Linguistics, 2020.

[ZWZ+19] Dongxiang Zhang, Lei Wang, Luming Zhang, Bing Tian Dai, and Heng Tao Shen, *The gap of semantic parsing: A survey on auto-*

matic math word problem solvers, IEEE transactions on pattern analysis and machine intelligence **42** (2019), no. 9, 2287–2305.