

Mapping probability word problems to executable representations

Simon Šuster^{★◇}

Jesse Davis[♣]

Pieter Fizez[♠]

Luc De Raedt[♠]

Pietro Totis[♣]

Walter Daelemans[♠]

Angelika Kimmig[♣]

[★]University of Melbourne

[♠]University of Antwerp

[♣]KU Leuven

[◇]simon.suster@unimelb.edu.au

Abstract

While solving math word problems automatically has received considerable attention in the NLP community, few works have addressed probability word problems specifically. In this paper, we employ and analyse various neural models for answering such word problems. In a two-step approach, the problem text is first mapped to a formal representation in a declarative language using a sequence-to-sequence model, and then the resulting representation is executed using a probabilistic programming system to provide the answer. Our best performing model incorporates general-domain contextualised word representations that were finetuned using transfer learning on another in-domain dataset. We also apply end-to-end models to this task, which bring out the importance of the two-step approach in obtaining correct solutions to probability problems.

1 Introduction

Solving math word problems automatically is an active area of research in natural language processing. It poses interesting challenges in extracting relevant entities and quantities from a concise textual narrative, and reasoning about the relationships between them to answer the question posed in the text. While several approaches for different areas of mathematics have been proposed in the past, the majority have focused on arithmetic and algebraic problems where the mathematical representation is specified as a simple equation system that can be used with a standard solver (Hosseini et al., 2014; Roy and Roth, 2015; Kushman et al., 2014; Koncel-Kedziorski et al., 2015; Upadhyay and Chang, 2015; Huang et al., 2016; Wang et al., 2017; Ling et al., 2017; Amini et al., 2019; Miao et al., 2020). This paper focuses on a class of word problems that have not received much attention: those about probability that may be found, for example, in introductory textbooks for discrete mathematics.

Problem text:

A complete cycle of a traffic light takes 80 seconds. During each cycle, the light is green for 40 seconds, amber for 10 seconds, and red for 30 seconds. At a randomly chosen time, what is the probability that the light will not be red?

Representation:

```
group(cycle).
size(cycle, 80).
property(property, [amber, green, red]).
given(exactly(10, cycle, amber)).
given(exactly(30, cycle, red)).
given(exactly(40, cycle, green)).
take(cycle, light, 1).
probability(none(light, red)).
```

Answer: 5/8

Table 1: An example from the NLP4PLP dataset (Dries et al., 2017): input text, intermediate representation, and correct answer.

Table 1 provides an example problem. The overall task, to which we adhere in this work, is to obtain the answer from the text, with the formal representation as an intermediate step. Generally, the representation contains one or more multisets of objects with certain properties, one or more actions that create new multisets, and a question that imposes a constraint on the result of the actions and asks for the probability of that constraint holding. In the example, we have one multiset or group, *cycle*, with size 80. The next four lines define a property with (at least) three mutually exclusive values (*amber*, *green*, *red*), and the number of objects in the multiset that have each of these values. The *take*-statement is an action that generates a new multiset (*light*) from the given one by taking 1 element (without replacement). The final line specifies the question by imposing that none of the taken elements has a property value *red*.

Automatically constructing such a representation from the natural language text is challenging for several reasons. The text may not explicitly state whether sampling is with or without replacement,

and this information may need to be inferred. Questions with co-referring expressions are common, and may contain noun ellipsis, e.g. “One is selected at random”, in which the antecedent is mentioned in a previous sentence. Furthermore, problems may require access to world knowledge, e.g. distinguishing between male and female names.

Compared to general word problems, the probability word problems have a number of unique characteristics. They typically mention several multisets (e.g. “a deck”) with the objects contained (e.g. “cards”), and the properties (e.g. “black”, “queen”) and sizes that the model needs to relate back to the corresponding objects. The problems rely heavily on Boolean operators to combine the objects’ properties, as well as specify constraints holding on these properties (e.g., by using quantifiers, such as “at least”). The Boolean operators and quantification feature much less prominently, or not at all, in other types of math word problems. As we show in section 6, these are particularly challenging in modelling, leaving ample opportunities for future work. A distinctive characteristic of probability problems is the need to construct intermediate sets with a sampling operation; the resulting sets need to be kept by the solving system and used when answering the final probability question. While predicates in several existing datasets of word problems allow a straightforward translation into arithmetic operators, probability word problems usually do not. This makes the case for modelling the intermediate meaning representation stronger.

We follow the approach of Dries et al. (2017), who solve such problems in two cognitively plausible steps. The first step transforms the text into a declarative representation of the problem itself, as in the example above. The second step uses a dedicated solver implemented in the probabilistic programming language ProbLog (De Raedt et al., 2007) to compute the final, numerical answer from the intermediate representation. In probability problems, an answer can be wrong even if its difference to the correct answer is numerically small, but such differences stand out more clearly on the problem specification level, making the latter a more appropriate modelling target.

Our main focus is on predicting the intermediate representation from the problem text, though we also consider models that directly map the text to the answer. To map the word problem texts to their executable representation, we focus here on neural

sequence-to-sequence (seq2seq) models. Our most important findings are:

- using a pretrained language model to encode the problem text into contextualised representations leads to superior performance on the largest available probability-problem dataset NLP4PLP (Dries et al., 2017), when compared to (i) the classical seq2seq model whose encoder uses non-contextualised word embeddings; (ii) end-to-end sequential approaches foregoing the solver; (iii) the rule-based system of Dries et al. (2017) and the simpler baselines,
- starting with the best seq2seq model that uses contextualised representations, in-domain transfer learning on the MATHQA dataset (Amini et al., 2019) containing a broader spectrum of math word problems leads to additional substantial improvements in the number of word problems answered correctly,
- although end-to-end problem answering can approximate the true probabilities, a two-step approach using the solver is crucial in obtaining solutions that are exactly correct.

Finally, we discuss the difficulties commonly encountered in solving probability word problems based on an analysis of errors.

2 Background

2.1 Solving general math word problems

The large majority of word problem solving approaches deal with relatively simple meaning representations, typically ranging from simple addition and subtraction in early (rule-based) research (Briars and Larkin, 1984; Fletcher, 1985; Bakman, 2007; Yuhui et al., 2010) to solving linear equations in later statistical-NLP approaches (Hosseini et al., 2014; Kushman et al., 2014; Koncel-Kedziorski et al., 2015; Roy and Roth, 2015; Upadhyay and Chang, 2015). Sequence-to-sequence neural models have established themselves as a general purpose framework that can be applied to a variety of math problems and formal representations. They have been shown to generate equations of which problem types do not exist in training data (Wang et al., 2017), and to scale well to generating longer sequences linearly, as in the case of Ling et al. (2017).

predicate	description
group	a fundamental set of objects
size	group’s cardinality
property	introduces an attribute and its values
given	provides the number of elements in a group having some property, either with numerical values, i.e. <code>given(exactly(n, group, property))</code> , or in terms of percentages/fractions, i.e. <code>given(exactly(rel(fraction, group), set, property))</code>
and, or, not	Boolean operators used to combine different properties. Boolean algebra laws apply, e.g. <i>and(red,king)</i> is equivalent to <i>and(king,red)</i> .
take, take_wr	a new set obtained by taking, respectively, with or without replacement n elements from a group
observe	defines a constraint holding on a set created by a take/take_wr action
probability	specifies a question in the problem; probability of an observable property of a set
at_least, exactly,	constraints on the properties of the objects in a set
all, some, ...	

Table 2: Summary of predicates used in the NLP4PLP language.

Among the works that adhere to the conceptual split between the text-to-representation mapping and solver application, like in our case, we can point out the following examples. Liang et al. (2016) present a hybrid rule-based and ML approach where the problem texts are first mapped to a linguistic representation that highlights the syntactic relationships between words, then a logic representation is built that is passed to the inference engine, which carries out math operations to obtain the answer. A similar decomposition of the problem has been previously studied in Matsuzaki et al. (2013), where the problems are translated to logical forms using Combinatory Categorical Grammar and Discourse Representation Structure formalisms, then rewritten into the input language of the solver.

2.2 Probability word problems

The works focusing specifically on probability word problems are scarce. We now discuss two largest datasets including such problems, NLP4PLP and MATHQA, and point to the differences between them.

NLP4PLP is the primary dataset used in our work, consisting of word problems in English annotated with declarative problem specifications (Dries et al., 2017). These contain a list of order-invariant statements, each consisting of predicates and arguments. The main types of predicates introducing the statements are listed in Table 2. MATHQA (Amini et al., 2019) is a recently introduced dataset which extends the AQUA dataset (Ling et al., 2017) with annotations of operations. It contains word problems in English from various math domains, including 663 on probability. These are annotated with formal representations of the steps needed to solve the

problem. There are two fundamental differences with the NLP4PLP dataset. First, MATHQA considers multiple choice questions, whereas NLP4PLP questions ask for a number. Second, MATHQA annotations describe *how* to solve the problem, whereas NLP4PLP describe *what* the problem is. The set of predicates in MATHQA defines the basic arithmetic operations as well as probability-specific operations. The latter include permutation and factorials, but they are applied infrequently in the dataset. The problem texts are similar in both datasets, whereas the annotations in MATHQA include only single (nested) statements which can be sequentially executed.¹ The authors apply neural seq2seq models and find that they improve over simpler baselines, but the gap to human performance remains large.

Another dataset that includes probability questions is the MATHEMATICS DATASET (Saxton et al., 2019). The questions are generated automatically, and are much more linguistically impoverished compared to ours. All included probability problems involve sampling without replacement from a bag of repeated letters,² and no formal representation of the problem is given. The main purpose of the dataset is to provide a testbed for analysing mathematical reasoning capabilities of neural models.

An early approach to solving probability questions is Gelb (1971). While Gelb’s high-level approach is similar to ours, the various components

¹E.g. *what is the probability that the sum of two dice will yield a 5, and then when both are thrown again, their sum will again yield a 5 ? assume that each die has 4 sides with faces numbered 1 to 4.* → `multiply(divide(4, power(4, const_2)), divide(4, power(4, const_2)))`

²E.g. *Three letters picked without replacement from qqkklkqkqk. Give prob of sequence qql.* → `1/110`.

are tackled in substantially different ways. It includes no learning in the NLP part, but uses a heuristic transformation-based approach to determine semantically-rich phrases in problem texts which can then be used by the solution generator. The latter then achieves a solution by constructing a combinatorial formula in a stepwise manner.

2.3 Mapping to other types of executable representations

There are similarities between our work and the existing NLP work on mapping text to other kinds of executable representations. In text-to-SQL mapping, the goal is to encode the database relations in an accessible way for the semantic parser, and to model the alignment between database columns and their mentions in a given query, which is analogous to our task. Seq2seq models with attention have been widely studied for this task as well (Zhong et al., 2017; Iyer et al., 2017). In addition to SQL representations, more general knowledge base representations (Zettlemoyer and Collins, 2005; Yih et al., 2015) and general-purpose source code generation for programming languages like Python and Java (Ling et al., 2016; Yin and Neubig, 2017) are conceptually similar as well.

3 Approaches to Solving Probability Word Problems

Our ultimate objective is to address the following:

Given: A natural language description of a word problem involving computing a probability.

Do: Produce the correct answer to the problem.

We will consider two high-level ways for addressing this task. The **two-step approach** attempts to encode the natural language description into the formal representation developed by Dries et al. (2017) specifically for representing probability problems. A dedicated solver exists for this representation. An advantage of this approach is that it promotes interpretability as the formal representation can be inspected.

The **end-to-end approach** simply attempts to train a deep architecture to directly predict the answer to the question based on the text. This bypasses the solver and hence circumvents the need to generate the formal representation of the problem. This is likely a more challenging problem. Moreover, it does not provide any way to check if the produced answer is correct.

We explored multiple ways to instantiate each approach, which we will now describe in more detail.

3.1 Two-step approach

We consider a baseline seq2seq approach and then explore how to augment it using contextualised representations from language models.

BiLSTM The baseline approach is a 400-dimensional, single-layer bidirectional LSTM encoder-decoder. The encoding is provided by processing the problem text as a sequence of tokens. The decoder then must generate the question’s representation in Dries et al.’s language as a linear sequence. The solver will use the output as is. Hence, the model must generate the structural parts of the representation such as punctuation and parentheses. We augment the textual input with three types of **additional features** obtained with the CoreNLP toolkit (Manning et al., 2014): part-of-speech tags of the entire problem sequence; numerical entities recognised by the NER component; and dependency label of the relation connecting the word to its parent.

We consider including three types of contextualised representations to the encoder:

FrozenEncoder uses the contextualised representations of the input text, but randomly initialises the BiLSTM encoder of our model and freezes it during training. Prior research indicates that using such random encoders can lead to performance that is robust and sometimes even competitive with finetuned encoders, since this approach maximally exploits the information present in the pretrained representations (Wieting and Kiela, 2019).

BERT is a transformer-based encoder that outputs context-dependent token activations. We use the pretrained uncased model.³

GPT-2 is another well-known transformer-based language model (Radford et al., 2019).⁴

For BERT and GPT-2 we consider two options: (i) a “frozen” variant that directly uses the provided representations as input for the decoder and (ii) a “finetuned” variant that is adapted by training on our corpora.

³<https://bit.ly/2SPUJQE>

⁴<http://huggingface.co/gpt2>

3.2 End-to-end models

We explore two approaches for directly predicting the answer to a probability problem based on the problem’s text.

Continuous-BiLSTM This is a regression model that encodes the text with a BiLSTM using the baseline architecture described in the previous section. It then predicts the final probability using a regression output layer consisting of single output node. The model is trained to minimise the squared error loss.

Discretised-BiLSTM Here, we treat the task as a multi-class classification problem where the probability space is segmented into k bins. The discretised approach simplifies the original problem which requires computing the exact probability to be given as answer. Varying $k \in \{2, 5, 10, 20\}$ enables comparing the network’s performance at different resolutions, although the smaller values are clearly less valuable due to oversimplification. Bin widths are chosen such that each bin contains an approximately equal percentage of total data. Again, the same encoder is used. The decoder now has a single softmax node and is trained using cross-entropy as the loss function.

4 Experimental setup

4.1 Data

We split the NLP4PLP dataset⁵ into training/development/test (80/10/10%) parts. The statistics of the dataset are reported in Table 3.

n train	1,720
n dev	217
n test	214
n tokens in problem texts	100,205
vocabulary size (lowercased)	4,686
avg. problem length (in tokens)	47

Table 3: Data statistics.

We pre-process the annotations in the following way. For numeric arguments, we do not use their actual values since the space is very large. Instead, we map the numbers to numbered symbols according to the order in which they occur in the text, e.g. “2 accidents in 1 year” $\rightarrow$ “ n_0 accidents in n_1 year”. The numbered symbols generated by the model are mapped back to the original numbers before executing the solver. Although we have also considered

using a pointer-based approach, the instances in our dataset generally do not contain indices for number locations in the problem text, while creating them automatically would be noisy.

4.2 Evaluation

We evaluate two aspects of the performance. The *surface* evaluation simply checks the correctness of the generated representation against the formal ground-truth representation. We report the accuracy (representations need to match exactly), as well as the F1 score, which also rewards the representations that are partially correct. The F1 score represents the average F1 over per-instance F1 scores; it measures the overlap between the generated and the ground-truth representations, which are split into tokens and treated as bags of words.⁶ The *execution-level* evaluation first passes the generated representation to the solver, then compares its output to that of the ground-truth answer (probability). We report the accuracy, in which the probabilities rounded to four decimal digits need to match exactly, as well as the mean-absolute error (MAE) to summarise the magnitude of prediction error.

4.3 Baselines

NearestNeighbour The nearest-neighbour baseline vectorises the problem text using pretrained 50-dimensional embeddings. Then, for test instance i , it finds the most similar training instance using:

$$\operatorname{argmax}_{n \in N} \cos\left(\frac{1}{|C_n|} \sum_{j \in C_n} c_j, \frac{1}{|Q|} \sum_{k \in Q} q_k\right), \quad (1)$$

where $c, q \in \mathbb{R}^d$, the multiset C_n contains all words in the training instance n , Q contains all words from the current test instance, and $\cos$ is the cosine similarity. In the two-step setting, the method returns the formal representation for the most similar training instance. In the end-to-end setting, the method predicts the probability that is the correct answer to the training instance.

Rule-based system Dries et al. (2017) present a semi-rule-based system that is specifically tailored to the NLP4PLP dataset. They focus on those word problem descriptions that mention groups of

⁶E.g. `size(cycle, 80) .-> {'size', '(', 'cycle', ',', '80', ')', '.'}`. In calculating the precision, the number of correct words is divided by the number of all predicted words. In recall, the former is divided by the number of words in the ground-truth representation.

⁵<https://bit.ly/33TYas2>

objects explicitly. The representations can only contain a single action and only represent a subset of the complete formal language.⁷ The approach of Dries et al. first PoS-tags and syntactically parses the dataset to extract numbers, which are linked to their respective entities by a multilayered perceptron. Using handcrafted rules, textual descriptions of the properties of entities are extracted from the parse trees. The problem question is identified with text search criteria, and then transformed using handcrafted rules into a structured form. A post-processing step removes any inconsistencies. The authors were able to apply the described approach to 41% cases from the entire dataset, of which 31% were solved correctly. When comparing our models to their system in Table 4, we take their test set predictions, and count those test instances that were not supported by their system (n=130) as incorrect.

Other baselines In the end-to-end approach, we also report the results of a random predictor (**Random**) that randomly samples from a uniform distribution $[0, 1)$, and those of a **MeanProbability** baseline which invariably predicts the mean probability as estimated on the training set.

4.4 Implementation details

When producing the formal intermediate representation during decoding, our BiLSTM model uses the additive attention mechanism (Bahdanau et al., 2014), and teacher forcing during training (Goodfellow et al., 2016). The BiLSTM models are trained for 30 epochs with the Adam optimiser (Kingma and Ba, 2014), and early-stopped based on the performance on the development set. We train and evaluate each model five times with different random seeds, and finally report the averaged performance.

Each additional linguistic feature is embedded into a 10-dimensional randomly-initialised vector. The BiLSTM models that use non-contextualised word representations use 50-dimensional word2vec embeddings (Mikolov et al., 2013) pretrained on Wikipedia and news corpora.

For the BiLSTM model, we consider two additional modifications to the testing regime. First, we use a **beam search** decoding that keeps the 10 best candidate outputs throughout the decoding process and then report the result for the best generated

representation. Second, as a variation on reporting the averaged performance over different runs, we can instead combine the predictions from different models by taking a **majority vote** after finding the best candidate representation using the beam search.

5 Results and discussion

5.1 Seq2seq models and the advantage of contextualised representations

The results for the two-step approach to solving probability word problems are shown in Table 4. Taking the BiLSTM model as the starting point of our discussion, we see that it accurately maps 19% of the problems to their ground-truth representation, with an F1 score of 0.88. This translates into execution-level accuracy of 0.32 for the cases among which a solution was found. When including also the representations with no solution, the corrected accuracy is 0.26. Here and throughout our results, we see that the execution accuracy surpasses that measured at the surface-level; this effect arises since some parts of the problems can be specified in alternative ways, which is penalised by the surface-level accuracy but not by the execution one. Introducing the beam decoding and majority-vote ensembling leads to further beneficial effects: the first boosts the execution-level accuracy by around 0.03 point, and the second, when coupled with beam search, brings the accuracy at execution time to around 0.37 (0.31 when discounting for solver errors). When including pre-trained contextualised representations, the GPT-2 model is competitive with our original BiLSTM encoder, while the BERT encoder is superior both when fine-tuned or frozen. The frozen BERT encoder clearly has the lowest MAE of all models, and as such contains the most easily exploitable domain-specific information.

The rule-based system of Dries et al. (2017) performs less well than the BiLSTM models, although it still has a clear advantage over the nearest-neighbour baseline and the randomly-initialised frozen encoder (Wieting and Kiela, 2019). In both baselines, no case can be exactly matched to the expected formal representation, and only one case was solved correctly (due to the leniency of the solver). The frozen encoder cannot provide meaningful information to the decoder, and confirms in our case that the BiLSTM encoder model effectively learns to encode task-specific information.

⁷“we excluded problems that are based on events (e.g., coin tosses), require observations [...], or that have aggregate or sequence constraints.”

Model	execution			surface	
	acc	acc _{disc}	MAE	acc.	F1
NearestNeighbour	0.005	0.005	0.317	0.0	0.773
Rule-based system	0.149	0.145	0.205	n/a	n/a
EncoderDecoder					
BiLSTM	0.318	0.262	0.212	0.187	0.882
BiLSTM (beam search)	0.352	0.290	0.210		
BiLSTM (majority vote)	0.367	0.308	0.208		
FrozenEncoder	0.005	0.005	0.277	0.0	0.612
BERT (finetuned)	0.424	0.327	0.194	0.220	0.934
BERT (frozen)	0.385	0.322	0.171	0.234	0.932
GPT-2 (finetuned)	0.330	0.290	0.203	0.187	0.918
GPT-2 (frozen)	0.220	0.178	0.240	0.136	0.906
Transfer learning					
MathQA-BERT (frozen)	0.376	0.327	0.190	0.215	0.921
MathQA-BERT (finetuned)	0.425	0.346	0.183	0.243	0.923

Table 4: Test set results, averaged over 5 runs for all models with random weight initialisation. The execution evaluation is based on solver outputs, with accuracy on instances not resulting in an error shown first, and the accuracy discounting for errors shown under *acc_{disc}*. The surface evaluation compares the generated problem definitions with the gold ones. For the rule-based system, we were unable to obtain its intermediate representations, so the surface evaluation scores are marked as “n/a”.

Furthermore, since the nearest-neighbour approach fails to map any case correctly, and yields the answers that are on average more than 0.3 points away from the true answer probability, this speaks in favour of the diversity of word problems included in the NLP4PLP dataset. Naturally, performing more complex matching between the testing and the training problems could allow for associating number and entities more precisely, and therefore lead to improvements over the simple approach included here.

Another observation about the results in Table 4 is that the models tend to score high in F1 even when their accuracy is low. This happens because large parts of the predicted representations still overlap with the gold ones, e.g. the names of predicates and the punctuation markers. For the best-performing models that use contextualised representations, all F1 scores exceed 0.9, in which case only small parts of the generated representations are expected to be incorrect. We shed more light on this in section 6.

Transfer learning on MATHQA While we have experimented with various pretrained encoders, none of them were in any way pretrained for our specific domain. As a first inquiry into the possible impact of domain-specific pretraining, we fine-tune the BERT encoder using the masked language

modelling objective on the raw unannotated data of MathQA (Amini et al., 2019), a large-scale dataset of math word problems. Such domain-specific but task-agnostic finetuning has proven effective for a wide range of NLP tasks and domains (Gururangan et al., 2020). In order to avoid overfitting, we train the BERT encoder for a single epoch on all 37,297 sentences of the MathQA dataset. The results in Table 4 show that finetuning this domain-adapted BERT encoder has a positive impact on the accuracy metrics, while the frozen encoder is less robust than the BERT encoder which was not domain-adapted. This demonstrates the potential of the approach while also highlighting the importance of the bias-variance tradeoff between general-domain and domain-specific pretraining.

5.2 End-to-end models

We now turn our discussion of the results to direct prediction of answer probabilities with the goal of discovering the effect of absence of intermediate representations and subsequent application of a dedicated solver. We see in Table 5 that our continuous-output neural model (BiLSTM regressor) beats all included baselines with a MAE of 0.2. This result is interesting since it represents a slight improvement even over the MAE score of the vanilla BiLSTM discussed among the two-step approaches (0.212; Table 4). However, a signif-

Model	acc.	MAE
Random	0.0	0.349
NearestNeighbour	0.0	0.307
MeanProbability	0.0	0.247
Continuous-BiLSTM	0.0	0.204
Discretised-Random ($k = 20$)	0.051	
Discretised-BiLSTM ($k = 20$)	0.160	
Discretised-Random ($k = 10$)	0.096	
Discretised-BiLSTM ($k = 10$)	0.232	
Discretised-Random ($k = 5$)	0.199	
Discretised-BiLSTM ($k = 5$)	0.369	
Discretised-Random ($k = 2$)	0.51	
Discretised-BiLSTM ($k = 2$)	0.677	

Table 5: End-to-end results on the test set, averaged over 5 runs. k represents the number of bins (classes) used when discretising the probability range. We omit the MAE scores when reporting the results for the discretised setting.

icant distinction is that the end-to-end regressor discussed here cannot find exactly correct solutions to *any* of the test problems. A conclusion we can draw is that—although the encoding part of the model is the same in both end-to-end and two-step approaches—modelling the intermediate representation appears to be a crucial step in arriving at exactly correct solutions.

An alternative approach to end-to-end modelling is to transform the problem answer space by discretising it into k bins. The BiLSTM classifier greatly improves over the random baseline for all values of k , but the absolute accuracies remain low for larger values of k . For example, when classifying with $k = 20$, the BiLSTM classifier correctly predicts three examples out of 20. While this represents a 200% improvement over the random baseline (which predicts correctly only one out of 20), a large majority of cases still remain incorrect even though we are already allowing simplified solutions that are not exactly correct but only approach the true answers.

6 Qualitative analysis

We base the error analysis on the investigation of the predicates in the gold representation from the test set that are not found in the predicted one. We classify an output with respect to the first type of predicate that does not find a correspondence in the predicted representation, according to the following order: group, size, given, take, observe, probability. We choose such or-

der because each predicate type relies on the information provided by the previous ones. In fact, errors usually result in incorrect predictions for the following types in the order as well. We take as source of our analysis the predictions of the best-performing model, i.e. MathQA-BERT (fine-tuned). Of the 214 predictions, 75 are correct and 139 present a missing statement on one of the aforementioned levels.

Broadly speaking, two types of errors stand out: i) those involving confusion about what kind of modifier or Boolean operator to choose, and ii) those involving extracting the right numeric argument for a predicate (e.g. how many items to sample using a `take` statement). We now analyse the errors in more detail.

Object sets and their cardinality There are 6 group statements that were not predicted (e.g. Error A1 in the appendix), 4 of them on problems with two different groups g_1 and g_2 with a fine-grained composition of the form `given(exactly(rel(..., g1))`: the network predicts only one group and links the composition of both sets to a single group. There are 10 problems with `size` statements not found in the corresponding prediction: 5 because the predicted size was wrong (e.g. Error A2) and 2 because the statement was missing at all. In both cases the following `given(exactly(...)` statements do not add up to the correct size.

Subset recognition based on a property The statements with `given` are the most difficult with 54 errors. 24 regard again statements of the form `given(exactly(rel(..., ...), ...))`: the network struggles to correctly identify all the subsets and their numerical relation with the whole group of objects (e.g. Error A3). In fact, other 5 predictions present errors related to sets defined in terms of union and intersection of existing groups (`and/or(..., ...)`). To deal with these errors, more complex syntactic features may be useful, e.g. features capturing coordination. 13 problems define the wrong number of objects in a given subset. Similar issues emerge from the predicates `take` (see Error A5): 7 of the 14 errors are due to a wrong number of taken objects and 4 due to taking from the wrong subset of a group defined (correctly) with `given(exactly(rel(..., ...))` statements. An alignment between the numeric

arguments and their position in the text could help. An observe statement was missing in 3 cases.

Question understanding Finally, the errors about probability correspond to cases that are almost correct (i.e. all previous statements are correct), and are mostly related to quantifiers. Of the 25 errors, 4 predict the wrong quantifier (all, exactly, atmost, ...), e.g. Error A6, 4 use the correct one on the wrong set, and 3 fail to correctly identify nested properties of the form `and/or (... , ...)` (e.g. Error A7).

Syntax errors Finally, we report 8 cases containing syntax errors, either due to unmatched parenthesis or unspecified arguments (e.g. `or ( , )`).

7 Conclusion

We have investigated the use of neural sequence-to-sequence models to generate intermediate representations for solving probability word problems, and shown the benefit of introducing contextualised word representations together with transfer learning on another dataset of math word problems. Our results also suggest that mapping to a problem specification followed by the application of a dedicated solver is preferable to end-to-end modelling where the answer probabilities are predicted directly from the encoded text. The qualitative analysis of results reveals that the extraction of relevant entities and quantities from concise textual descriptions, as well as reasoning about the relationships between them are still challenging, and therefore provide possible directions for future work.

Acknowledgments

We would like to thank the anonymous reviewers for their feedback. This research was carried out in the framework of the FWO-Vlaanderen project n. G066818N.

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. [MathQA: Towards interpretable math word problem solving with operation-based formalisms](#). In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2357–2367, Minneapolis, Minnesota. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Yefim Bakman. 2007. Robust understanding of word problems with extraneous information. *arXiv preprint math/0701393*.

Diane J Briars and Jill H Larkin. 1984. An integrated model of skill in solving elementary word problems. *Cognition and instruction*, 1(3):245–296.

Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. 2007. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI’07*, page 2468–2473, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Anton Dries, Angelika Kimmig, Jesse Davis, Vaishak Belle, and Luc De Raedt. 2017. Solving probability problems in natural language. In *Proceedings Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 3981–3987.

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

Jack P. Gelb. 1971. Experiments with a natural language problem-solving system. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence, IJCAI’71*, page 455–462, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*, volume 1. MIT press Cambridge.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. [Don’t stop pretraining: Adapt language models to domains and tasks](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online. Association for Computational Linguistics.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. [Learning to solve arithmetic word problems with verb categorization](#). In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533, Doha, Qatar. Association for Computational Linguistics.

Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. 2016. [How well do computers solve math word problems? large-scale dataset construction and evaluation](#). In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 887–896, Berlin, Germany. Association for Computational Linguistics.

- Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. [Learning a neural semantic parser from user feedback](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, Vancouver, Canada. Association for Computational Linguistics.
- Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. [Parsing algebraic word problems into equations](#). *Transactions of the Association for Computational Linguistics*, 3:585–597.
- Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. [Learning to automatically solve algebra word problems](#). In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 271–281, Baltimore, Maryland. Association for Computational Linguistics.
- Chao-Chun Liang, Kuang-Yi Hsu, Chien-Tsung Huang, Chung-Min Li, Shen-Yu Miao, and Keh-Yih Su. 2016. [A tag-based English math word problem solver with understanding, reasoning and explanation](#). In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 67–71, San Diego, California. Association for Computational Linguistics.
- Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Fumin Wang, and Andrew Senior. 2016. [Latent predictor networks for code generation](#). In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 599–609, Berlin, Germany. Association for Computational Linguistics.
- Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. [Program induction by rationale generation: Learning to solve and explain algebraic word problems](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167, Vancouver, Canada. Association for Computational Linguistics.
- Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. [The Stanford CoreNLP natural language processing toolkit](#). In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland. Association for Computational Linguistics.
- Takuya Matsuzaki, Hidenao Iwane, Hirokazu Anai, and Noriko Arai. 2013. [The complexity of math problems – linguistic, or computational?](#) In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*, pages 73–81, Nagoya, Japan. Asian Federation of Natural Language Processing.
- Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. [A diverse corpus for evaluating and developing English math word problem solvers](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online. Association for Computational Linguistics.
- Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *ICLR*.
- Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.
- Subhro Roy and Dan Roth. 2015. [Solving general arithmetic word problems](#). In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752, Lisbon, Portugal. Association for Computational Linguistics.
- David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. In *ICLR*.
- Shyam Upadhyay and Ming-Wei Chang. 2015. Draw: A challenging and diverse algebra word problem set. Technical report, Number MSR-TR-2015-78, Oct. 2015.[Online]. Available: <https://www.microsoft.com/...>
- Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. [Deep neural solver for math word problems](#). In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854, Copenhagen, Denmark. Association for Computational Linguistics.
- John Wieting and Douwe Kiela. 2019. [No training required: Exploring random encoders for sentence classification](#). In *International Conference on Learning Representations*.
- Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. [Semantic parsing via staged query graph generation: Question answering with knowledge base](#). In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1321–1331, Beijing, China. Association for Computational Linguistics.
- Pengcheng Yin and Graham Neubig. 2017. [A syntactic neural model for general-purpose code generation](#). In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada. Association for Computational Linguistics.

- Ma Yuhui, Zhou Ying, Cui Guangzuo, Ren Yun, and Huang Ronghuai. 2010. Frame-based calculus of solving arithmetic multi-step addition and subtraction word problems. In *2010 Second International Workshop on Education Technology and Computer Science*, volume 2, pages 476–479. IEEE.
- Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence, UAI’05*, page 658–666, Arlington, Virginia, USA. AUAI Press.
- Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

A Appendix: error examples

A1) Error type: missing group (Example I63)

The drama school is getting ready for the play. Since the play is an exercise in learning, a random drawing will take place to assign parts. There are 37 boys and 51 girls in the school. There are 35 parts in the play for boys and 42 parts in the play for girls. Dominique is hoping for the female lead role. 5 female roles have been chosen and the lead female role is still available. What is probability that Dominique will land the role now?

Gold	Predicted
<pre>group(girls) . size(girls, 51) . group(rest(roles)) . given(exactly(1, girls, lead)) . take(rest(roles), dominique, 1) . take(girls, roles, 5) . observe(none(girls, lead)) . probability(all(dominique, lead)) . property(property0, [lead]) .</pre>	<pre>group(girls) . size(girls, 51) . given(exactly(1, girls, lead)) . take(girls, dominique, 1) . probability(all(dominique, lead)) . property(property0, [lead]) .</pre>

A2) Error type: wrong size (Example I15)

Half the face of a fair die are painted blue, half yellow. The die is rolled twice. What is the probability the die will turn up blue both times?

Gold	Predicted
<pre>group(die) . size(die, 6) . given(exactly(1, die, blue)) . given(exactly(1, die, yellow)) . take_wr(die, rolled, 2) . probability(all(rolled, blue)) . property(property0, [blue, yellow]) .</pre>	<pre>group(die) . size(die, 2) . given(exactly(1, die, blue)) . given(exactly(1, die, yellow)) . take_wr(die, rolled, 2) . probability(all(rolled, blue)) . property(property0, [blue, yellow]) .</pre>

A3) Error type: wrong rel (Example h595)

A man finds that on the average he hits the target 9 times out of every 10 times and scores a bull's eye on the average once every 5 rounds. He fires 4 rounds. What is the probability that he scores at least 2 bull's eyes?

Gold	Predicted
<pre>group(scores). given(exactly(rel('/',(1,5),scores), scores,bull)). take_wr(scores,round,4). probability(atleast(2,round,bull)). property(property0,[bull]).</pre>	<pre>group(scores). given(exactly(rel('/',(9,10),scores), scores,bull)). take_wr(scores,round,5). probability(atleast(2,round,bull)). property(property0,[bull]).</pre>

A4) Error type: wrong subgroup size (Example l1137)

A jar contains 6 blue marbles, 12 green marbles and 7 yellow marbles. Find the probability of randomly drawing a green marble.

Gold	Predicted
<pre>group(jar). given(exactly(6,jar,blue)). given(exactly(12,jar,green)). given(exactly(7,jar,yellow)). take(jar,marble,1). probability(all(marble,green)). property(property0,[blue,green,yellow]).</pre>	<pre>group(jar). given(exactly(12,jar,blue)). given(exactly(7,jar,green)). given(exactly(12,jar,yellow)). take(jar,marble,1). probability(all(marble,green)). property(property0,[blue,green,yellow]).</pre>

A5) Error type: wrong take (Example l847)

In a hurdle race, a player has to cross 10 hurdles. The probability that he will clear each hurdle is 5/6. What is the probability that he will knock down fewer than 2 hurdles?

Gold	Predicted
<pre>group(hurdles). given(exactly(rel('/',(5,6),hurdles), hurdles,clear)). take_wr(hurdles,cross,10). probability(less.than(2,cross,not(clear))). property(property0,[clear]).</pre>	<pre>group(hurdles). given(exactly(rel('/',(5,6),hurdles), hurdles,clear)). take_wr(hurdles,cross,4000). probability(atmost(2,cross,clear)). property(property0,[clear]).</pre>

A6) Error type: wrong quantifier (Example l228)

Two cards are drawn at random from a deck of 52 cards without replacement. There are 4 Kings. What is the probability that exactly one is a King, given that at most one is a King?

Gold	Predicted
<pre>group(deck) . size(deck, 52) . given(exactly(4, deck, king)) . take(deck, cards, 2) . observe(atmost(1, cards, king)) . probability(exactly(1, cards, king)) . property(property0, [king]) .</pre>	<pre>group(deck) . size(deck, 52) . given(exactly(4, deck, king)) . take(deck, cards, 2) . observe(atmost(1, cards, king)) . probability(atleast(1, cards, king)) . property(property0, [king]) .</pre>

A7) Error type: wrong logic (Example l794)

A couple have three children. What is the probability that among the children, there will be at least one boy or at least one girl?

Gold	Predicted
<pre>group(children) . size(children, 2) . given(exactly(1, children, boy)) . given(exactly(1, children, girl)) . take_wr(children, couple, 3) . probability(or(atleast(1, couple, boy), atleast(1, couple, girl))) . property(property0, [boy, girl]) .</pre>	<pre>group(children) . size(children, 2) . given(exactly(1, children, boy)) . given(exactly(1, children, girl)) . take_wr(children, couple, 3) . probability(atleast(1, couple, boy)) . property(property0, [boy, girl]) .</pre>