Discrete Reasoning Templates for Natural Language Understanding

Hadeel Al-Negheimish **Pranava Madhyastha** **Alessandra Russo**

Department of Computing
Imperial College London
{halnegheimish,pranava,a.russo}@imperial.ac.uk

Abstract

Reasoning about information from multiple parts of a passage to derive an answer is an open challenge for reading-comprehension models. In this paper, we present an approach that reasons about complex questions by decomposing them to simpler subquestions that can take advantage of single-span extraction reading-comprehension models, and derives the final answer according to instructions in a predefined reasoning template. We focus on subtraction based arithmetic questions and evaluate our approach on a subset of the DROP dataset. We show that our approach is competitive with the state of the art while being interpretable and requires little supervision.

1 Introduction

Automated reading comprehension (RC) is an important natural language understanding task, where a model is presented with a passage and is asked to answer questions about that passage. While models have excelled at single-span extraction questions, they still struggle with reasoning over distinct parts of a passage (Dua et al., 2019). Several multi-hop reasoning benchmarks have been proposed (Yang et al., 2018; Khashabi et al., 2018; Dua et al., 2019), of which, in this paper, we focus on the DROP (Discrete Reasoning Over the content of Paragraphs) dataset. Inspired by the semantic parsing literature, the dataset contains questions that involve possibly multiple steps of discrete reasoning over the contents of paragraphs, including numerical reasoning.

Recent work has proposed several novel approaches to tackle DROP (Ran et al., 2019; Hu et al., 2019; Andor et al., 2019; Gupta et al., 2020; Chen et al., 2020). However, most approaches provide little evidence of their reasoning process, especially with regards to *why* specific operands are chosen for a reasoning task. With the exception of (Gupta et al., 2020; Chen et al., 2020), they also suffer from limited compositionality.

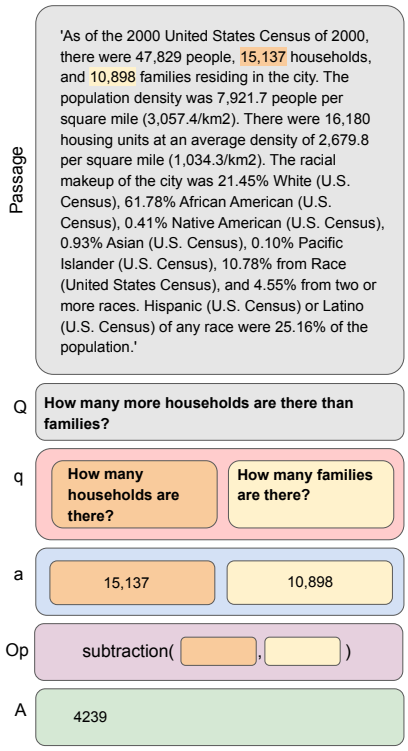

Figure 1: Subtraction Template: Original question is decomposed to two simpler subquestions that find the values associated with the two compared entities as span-extraction, and the final answer is calculated by finding the absolute difference of the two partial answers.

In this paper, we present a first attempt at building an interface between discrete reasoning and unstructured natural language. We propose decomposing a question to simpler subquestions that can more easily be solved by single-span extraction RC models. Such decomposition is defined by Reasoning Templates, which also determine how to assemble the computed partial answers. We demonstrate the feasibility of our approach with the *subtraction* based questions (illustrated in Figure 1). We show that our approach is competitive with the state of the art models on a subset of DROP's subtraction

Proceedings of the 16th Conference of the European Chapter of the Associationfor Computational Linguistics: Student Research Workshop, pages 80–87
April 19 - 23, 2021.

questions while requiring much less training data and providing visibility of the model's decision-making process.

2 Related Work

There has been a recent resurgence in research on automated reading comprehension (RC) where an automated system is capable of reading a document in order to answer the questions pertaining to the document. This has led to the creation of several RC datasets to facilitate the research (Rajpurkar et al., 2016, 2018; Yang et al., 2018; Reddy et al., 2019; Dua et al., 2019; Huang et al., 2019). Among these, SQuAD (Rajpurkar et al., 2016) is a popular *single-hop question answering* dataset where a question can be answered by relying on a single sentence from the document. SoTA models have achieved near-human performance on such single-hop question answering tasks.[1] However, answering a question by only identifying the most relevant span leaves models prone to exploiting advanced pattern matching algorithms.

On the other hand, multi-hop questions make reading-comprehension more challenging, as they require integrating information from multiple parts in a passage (Yang et al., 2018; Khashabi et al., 2018; Dua et al., 2019). DROP (Dua et al., 2019) is one such dataset that contains questions covering many types of reasoning, such as counting, sorting, or arithmetic. The dataset was constructed by adversarially crowdsourcing questions on a set of Wikipedia passages known to have many numbers. Special model architectures have been built to tackle DROP, and these fall into two general directions: the first direction augments reading comprehension models that were successful on single-span extraction questions with specialized modules that tackle more complex questions. These include NAQANet (Dua et al., 2019), NumNet (Ran et al., 2019) , and MTMSN (Hu et al., 2019). The second direction works on predicting programs that would solve the question, CalBERT (Andor et al., 2019) defines a set of derivations and scoring functions for each of them, while more recent work NMN (Gupta et al., 2020) and NeRd (Chen et al., 2020) utilize LSTMs to decode variable-length programs from question and passage embeddings.

By definition, models with specialized modules have limited compositional reasoning abilities. The two directions vary in their interpretability; the first shows which module has been used, and the second shows the resulting programs which have been generated to compute the answer. However, none of these directions indicate why operands in the passage were selected. For all approaches, the dataset is augmented with all possible derivations that lead to the gold answer, by performing an exhaustive search. Moreover, all approaches assume a preprocessing step that extracts all numbers in the passage and their indices, which massively reduces the search space for arithmetic questions.

In this work, we build upon DecompRC (Min et al., 2019) for question decomposition, where a model is trained to extract key parts of content from the question which are then used for decomposition. Arithmetic questions, which we focus on in this work, are a known limitation of DecompRC. An alternative approach to decomposition is QDMR (Wolfson et al., 2020), a recently proposed formalism for decomposing questions into a series of simpler steps based on predefined query operators. QDMR breaks down a question to its atomic parts directly, whereas we propose recursively decomposing questions to simpler ones. While (Wolfson et al., 2020) provides a dataset of annotated questions, QDMR parsing remains an open challenge. In the following section we present our approach that focuses on answering arithmetic questions.

3 Approach

We propose a pipelined approach that focuses on breaking down complex questions that require reasoning over multiple parts in the passage to simpler single-hop questions. The latter can be resolved by taking advantage of state of the art single-hop reading comprehension models. The main building block of our approach is a *reasoning template*. Each reasoning type is associated with a single template, which contains instructions on how to decompose a question and how to combine partial answers to arrive at the final answer.

Figure 2 illustrates our pipeline. First, the question and passage are fed to our system, which selects a template depending on the reasoning type required (classification task). The template decomposes the question to simpler subquestions that are then passed on to a single-hop RC model. Partial answers are used to arrive at an answer according to the instructions provided by the template. Some questions need further decomposition, and the appropriate template will be chosen for the sub-

[1] https://rajpurkar.github.io/SQuAD-explorer/

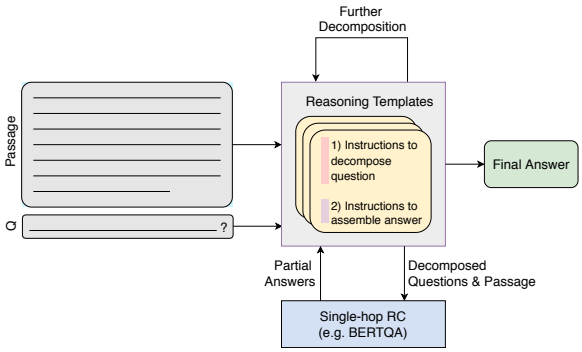

Figure 2: Model Overview: Given a question, decompose it into simpler questions according to a template such that they can be answered by a single-hop RC model, and assemble the final answer by applying the operation associated with the template.

question.[2]

For the question decomposition component, our approach closely follows and builds upon DecompRC (Min et al., 2019), originally proposed for multihop, multidocument question answering. We repurpose the model for multihop arithmetic questions. DecompRC uses a two step approach to decompose questions. First, a pointer model is trained to identify a key part of the question that is used to formulate the sub-questions. Second, the predicted pointers are used to procedurally change the original question into two or three sub-questions. The approach is defined for three types of questions: Bridging, Intersection, and Comparison; each of them uses a different pointer model and a different heuristic procedure to generate the sub-questions.

In this paper, we propose the discrete reasoning template framework and demonstrate its potential by defining a single template: *subtraction*. We describe in detail our approach in the following subsections.

3.1 Question Decomposition

Question decomposition is a two-step process that includes identifying relevant information in the text of the original question (span extraction), and then using those spans for heuristic generation of sub-questions. The output of this step are simpler sub-questions, see 'q' in Figure 1.

Span Extraction For subtraction questions, the spans we are interested in identify two entities whose associated values are to be subtracted. Consider '$_0$How $_1$many $_2$more $_3$*households* $_4$are $_5$there $_6$than $_7$*married* $_8$*couples* $_9$*living* $_{10}$*together*$_{11}$?'. We need to extract the start and end indices of the first entity *households*, and the start and end indices of the second entity *married couples living together*, which are $[3, 3, 7, 10]$.

The pointer model is trained to predict 4 pointers, the start and end indices of the first and second entity respectively. Concretely, the model extracts 4 indices, $p_1 \leq p_2 \leq p_3 \leq p_4$, that surround the two spans of interest, maximizing the joint probability:

$$p_1, \cdots, p_4 = \underset{\{i_1 \leq \cdots \leq i_4\}}{\arg\max} \prod_{j=1}^{4} \mathbb{P}(i_j = ind_j)$$

where $\mathbb{P}(i_j = ind_j) = Y_{ij}$ is the probability that the ith word is the jth index produced by the pointer, and

$$Y = \text{softmax}(UW) \in \mathbb{R}^{n \times 4}$$

where W is a learned weight matrix of size $h \times 4$ and U is the contextualized embeddings of length h produced by pre-trained BERT(Devlin et al., 2019) of the n tokens in the original question:

$$U = \text{BERT}(S) \in \mathbb{R}^{n \times h}$$

We then train this model using cross entropy loss until convergence.

Subquestion Generation We find that the sub-questions needed in our approach have a high degree of overlap with the original question, making them amenable to heuristic decomposition as in DecompRC (Min et al., 2019). While DecompRC is defined for bridging, intersection and comparison type questions, we extend it with a separate procedure to handle *subtraction* type questions as described below. We outline in Algorithm 1 how subquestions can be generated for subtraction questions, given the pointers that have been predicted by the previous step. The algorithm keeps words that are common for both subquestions and then places each of the entities in the center of the generated questions. First, we chunk the original question into parts using the pointers as in lines 2-6. In lines 7-9, we remove comparative adjectives and adverbs from the first part. Before concatenating the different parts again, we remove the extra words from the middle part, utilizing the dependency parse of the original question.

[2]Further decomposition and template selection modules are left for future research.

Algorithm 1: Subquestion generation for subtraction questions

Data: Original question q: string, pointers P_4: array of length 4
Result: subquestions q_1, q_2: strings

```
dep_parse= dependency_parse(q);
part1 ← q[0:p1];
ent1←q[p1:p2 + 1];
middle←q[p2 + 1:p3];
ent2←q[p3:p4 + 1];
part2← q[p4 + 1:end];
for word in part1 do
    if word.pos_tag in ['JJR', 'RBR'] then
        remove word from part1;
head ← dep_parse.parent(ent2);
i ← head.i;
prev_i ← i;
while (head in middle) AND (prev_i-i ≤ 1) do
    new_head ← dep_parse.parent(head) ;
    remove head from middle;
    head ← new_head;
    prev_i ← i;
    i ← head.i;
q1← part1+ent1+middle+part2;
q2← part1+ent2+middle+part2;
```

3.2 Single-hop question answering

Once we have decomposed questions into simpler, single-hop questions, we can use the subquestions to extract the appropriate operands for reasoning from the passage. We opt to make use of a pre-trained off-the-shelf single-span extraction model, details provided in section 4. This is one possible instantiation for the model, and we can use any robust span-extraction model in its place.

3.3 Operation

A reasoning template includes instructions on how to perform two main steps; the first step decomposes a question to simpler subquestions as we have described in section 3.1. The second step, operation, is designed to derive the final answer given partial answers to decomposed questions. In the case of subtraction, it is simply the absolute difference between the two retrieved values, see '*Op*' in Figure 1. In the case where a span retrieved for a decomposed question contains more than a single number, we use the first number in the span.

4 Experiments

We start with a single template to demonstrate our approach: *subtraction*. Subtraction questions rely on finding the difference between two numbers to find the answer, they are usually in the form of *'How many more..?'* or *'How many fewer..?'*.

Dataset For evaluation, we collect two sets of subtraction questions from the DROP development set. The first, *clean*, is a subset of 52 questions curated by filtering the original dataset to find questions that contain words with 'JJR' or 'RBR' postags (comparative adjective and comparative adverb respectively), and from those we randomly sample 10 questions at a time and manually identify subtraction questions. We also annotate each of these questions with gold decompositions, two subquestions for each complex question. The other evaluation set, *noisy*, is a larger dataset that has been heuristically generated, this is intended to support generalizability of results on the smaller evaluation set. It contains 892 questions that have been filtered using trigrams at the beginning of the question: *'How many more'* or *'How many fewer'*.

There are two learning components in our pipeline: a pointer model to extract relevant entities from the question and a single-hop RC model to answer decomposed questions. For the latter, we use an off-the-shelf pre-trained BERT (Devlin et al., 2019) question answering model, which has been fine-tuned on SQuAD (Rajpurkar et al., 2016), a single-hop reading comprehension dataset. Specifically, we use the one provided by the huggingface transformer library (Wolf et al., 2020). As for the former, to train the pointer model we follow (Min et al., 2019) and annotate 200 examples. The data for this was gathered from the DROP training set in the same way we curated the *clean* evaluation set, for this step we simply identify the compared entities and delimit them with '#'.

4.1 Results and Discussion

Evaluating Question Decomposition In Table 1 we report the accuracy of the pointer model on the *clean* subtraction evaluation set, and in Table 2 we measure the overlap between the resulting spans and the annotated entities. While getting all pointers to match label succeeds for 73% of the data, we note that the accuracy of each of the pointers is much higher. We find that the pointer delimiting the start of the first entity is seemingly the most difficult to predict, which is also seen in lower F1 score for the first entity. We conjecture this to be the likely case as the second entity is usually preceded by words such as 'than' or 'compared to'.

We also measure the similarity between decomposed questions generated by our approach and the manually annotated gold decompositions. Table 3

	p1	p2	p3	p4	all
Acc	84.0 ± 0.9	88.5 ± 1.6	98.1	94.9 ± 0.9	73.1

Table 1: Accuracy of Pointer_4 model, we list the accuracy of individual pointers separately and accuracy of all pointers for each example. Results are reported as an average of 3 runs of the model with different random seeds.

	First Entity	Second Entity
F1	0.89 ± 0.02	0.97 ± 0.003
Precision	0.91 ± 0.018	0.96
Recall	0.90 ± 0.023	0.99 ± 0.006

Table 2: Measured overlap between resulting spans of the predicted pointers and the annotated entities, averaged over all questions in *clean* evaluation set.

displays the Word Mover's Distance metric (Kusner et al., 2015) and cosine similarity, based on the GloVe word embeddings shipped with SpaCy's (Honnibal et al., 2020) `en_core_web_lg` model. For most questions, the two subquestions match perfectly between the gold annotations and the generated ones. However, upon manual inspection, we find that the generated subquestion might sometimes omit the final verb. This is because of our traversal of the dependency parse in Algorithm 1. We found BERTQA was robust to these differences when extracting the related span from the passage.

Evaluating the Approach Table 4 shows the accuracy of each of the models on the subtraction evaluation sets. Since the result is a number, accuracy is evaluated as an exact match between the predicted answer and its label. We compare our approach with the state-of-the-art; MTMSN (Hu et al., 2019), the best performing model with specialized modules; and NeRd (Chen et al., 2020), the most recent work based on program induction. These were evaluated on the original questions in subtraction evaluation set. For our work, we evaluate two different variations: We run the pipeline on the gold decompositions that have been manually rewritten, and automatically-decomposed questions generated by our approach, using BERT single-hop RC described in section 4. For both gold-decompositions and learned-decompositions we get promising results that are on par with the state-of-the-art on this dataset.

When investigating the mistakes that our approach makes on the *clean* set, we find that many

Similarity Measure	q1	q2
WMD_{max}	3.56	4.43
WMD_{avg}	0.2266	0.6714
WMD_{median}	0.0	0.0
$cos(\theta)_{min}$	0.9538	0.9476
$cos(\theta)_{avg}$	0.9959	0.9913
$cos(\theta)_{median}$	1.0	1.0

Table 3: Reported similarities between manually decomposed questions (gold) and decompositions generated by our approach. We use word mover's distance (WMD) and cosine similarity of average word embeddings. For the former we report *max* distance, while in the latter we report *min* similarity as these highlight the worst-case of all subquestions. For most examples, the gold decompositions and generated subquestions overlap perfectly, as indicated by *median* score.

	Model	Acc_c	Acc_c^-	# MM	Acc_n
SoTA	MTMSN	86.5	89.4	3	81.3
	NeRd	73	76.6	2	62.3
Ours	Decomp_G	78.8	85.1	1	-
	Decomp_L	74.4 ± 2.4	79.9 ± 2.6	1	64

Table 4: Accuracy of models for subtraction questions. We report accuracy on *clean* evaluation set (52 questions) in Acc_c, accuracy after omitting 5 mislabeled questions in the second column (Acc_c^-) and specify how many of these Mislabeled questions Match the prediction in the #MM. The last column (Acc_n) reports accuracy on the *noisy* evaluation set (892 questions). Learned Decompositions (Decomp_L) are averaged over 3 random seeds in pointer model training.

of the mistakes are actually due to incorrect labels. The gold answer (or label) does not match the correct answer for a certain question. To validate this, we check the entire *clean* evaluation set and manually label each question. We find that 5 of the 52 questions are incorrectly annotated, one of these questions is actually invalid as the information needed to answer it does not exist in the passage. To better understand the effect of this, we discard incorrectly labeled examples and report accuracy in the second column of Table 4. We also report the number of predictions matching the incorrect label. The primary set of *true* mistakes our model makes are due to some questions needing further decomposition, eliciting commonsense knowledge, or because they are not *subtraction* questions, i.e. can be classified as MTMSN's `Negation` rather than `Diff` module.

NeRd fails on 3 questions that MTMSN and our approach got correctly because it could not produce a valid program to be evaluated. It also failed on 2 of the `Negation` question that our approach failed on, not because it was not able to address those kinds of question, but because the attention mechanism ignored a condition in the question "18 or over". Surprisingly, NeRd failed on both questions that necessitate nested processing, even though the architecture allows for compositionality. The remaining failure cases are due to choosing incorrect operands for the difference, but it is not clear why the model made those choices.

Discussion We find that our approach is promising; it is interpretable and requires little training data when compared to previous approaches, without compromising performance. Steps to arrive at an answer are explicit, and we can interpret each of the retrieved operands by their associated subquestions. Figure 1 shows an example of this for subtraction questions. MTMSN indicates which module was used, but it does not show what led to this particular choice of the arithmetic expression. Likewise, NeRD shows the program necessary to find the answer, but there is no indication on why the operands of each function were chosen.

The only training data needed was a small subset (200 examples) to train the pointer model, and in the future we need some data to train reasoning type classifier and other templates' pointer models. This comes in contrast to the exhaustive search needed to find all possible derivations to reach an answer for all questions in the training set (77.4k examples). Reasoning Templates retrieve operands for the subtraction operation by answering subquestions that refer to a particular number, making it more robust to noise in the annotation. We started by focusing on the subtraction template, because it is the most prevalent numerical reasoning type (with an estimated proportion of 29% of all questions (Dua et al., 2019)). However, this approach can be similarly extended to other reasoning types by defining a template for each, such as *date-difference* or *addition*.

We believe that such reasoning templates would be able to answer compositional questions with its recursive *decomposition* component. While this exploration is left for future research, we believe it is useful it outline how we expect it to handle compositionality. Recall from Figure 1 that input questions are passed to a classifier that selects which template to apply, one of the classes decides if the question is single-span and should be passed on to single-hop RC directly. Decomposed questions should also be passed through this classifier to determine if they need further decomposition.

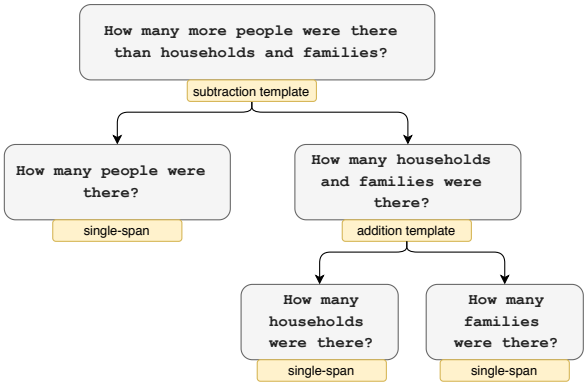

Figure 3: An example of how questions are further decomposed to facilitate compositionality.

Figure 3 shows an example where the second subquestion involves another reasoning task. After further decomposing it to single-span extraction questions and finding the solution to the addition operation, that answer would be passed to the previous task. This recursive processing should ideally allow for compositionality.

After building the entire pipeline we expect mistakes like nested operations and mis-classified `Negation` types to be rectified, boosting performance further. One challenge we wish to overcome is the engineering bottleneck involved in crafting each of the templates. Future work would explore methods that learn to construct these the templates.

5 Conclusion

We propose using Reasoning Templates for tackling reading comprehension tasks that involve reasoning over multiple paragraphs. We show that this approach is competitive with state of the art models on a subset of DROP's subtraction questions, while requiring much less training data and providing better visibility of the model's decision making. In future work, we plan on extending to further templates and investigate how to learn templates instead of working from a predefined set.

Acknowledgements

This research has been supported by a scholarship from King Saud University. We thank our anonymous mentor and reviewers for their constructive comments and suggestions.

References

Daniel Andor, Luheng He, Kenton Lee, and Emily Pitler. 2019. Giving BERT a calculator: Finding operations and arguments with reading comprehension. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5947–5952, Hong Kong, China. Association for Computational Linguistics.

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V. Le. 2020. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *International Conference on Learning Representations*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proc. of NAACL*.

Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. 2020. Neural module networks for reasoning over text. In *International Conference on Learning Representations*.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength Natural Language Processing in Python.

Minghao Hu, Yuxing Peng, Zhen Huang, and Dongsheng Li. 2019. A multi-type multi-span network for reading comprehension that requires discrete reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1596–1606, Hong Kong, China. Association for Computational Linguistics.

Lifu Huang, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Cosmos QA: Machine reading comprehension with contextual commonsense reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2391–2401, Hong Kong, China. Association for Computational Linguistics.

Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. 2018. Looking beyond the surface: A challenge set for reading comprehension over multiple sentences. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 252–262, New Orleans, Louisiana. Association for Computational Linguistics.

Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. 2015. From word embeddings to document distances. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 957–966. JMLR.org.

Sewon Min, Victor Zhong, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2019. Multi-hop reading comprehension through question decomposition and rescoring. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6097–6109, Florence, Italy. Association for Computational Linguistics.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. 2019. NumNet: Machine reading comprehension with numerical reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2474–2484, Hong Kong, China. Association for Computational Linguistics.

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2019. CoQA: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266.

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. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. 2020. Break it down: A question understanding benchmark. *Transactions of the Association for Computational Linguistics*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

A Experimental Settings

A.1 Model Settings

We use the final layer of $\text{BERT}_{\text{LARGE}}$ (Devlin et al., 2019) to produce contextualized embeddings used for span extraction fine-tuned to extract 4 pointers.[3] We use Adam optimizer with learning rate of $5e-5$ and warm-up over the first 10% steps to train. Loss function is calculated with cross-entropy. Training batch size is 20 examples. We train three models with different random seeds and report average performance over these.

A.2 SoTA Comparison

We report the accuracy of MTMSN (Hu et al., 2019) and NeRd (Chen et al., 2020) on the two subtraction evaluation sets in Table 4. For MTMSN, we use the pre-trained `MTMSN_LARGE` model published on their github page. Code and model checkpoints for NeRd were shared in email communication with the authors in June, 2020.

B Evaluating on a larger dataset

We started evaluating this work on the smaller, *clean*, dataset of 52 questions that has been manually curated. To validate that this sample is representative of subtraction questions in the DROP devset, we worked to heuristically identify relevant questions. We started with the same 2 steps involved in the manual curation, filter the devset (9536 questions) for questions that have 'number' as answer type (leaves 5850 questions) and contain comparative adjectives or adverbs. This leaves us with a subset of 1386 questions. The above conditions cover more questions than we are interested in, e.g. 'How many people are 18 or older?'. We refine the second condition to exclude sentences where the `JJR|RBR` tokens are preceded with an *or*, this omits 146 more samples. We proceed by passing these through our pipeline. Below is a summary of failure cases of the different components of our approach:

a. 1 sample did not produce valid pointers (used [SEP] token which is BERT-specific).

b. 22 samples did not produce valid decomposition. This is due to issues in mismatching tokenization between the pointer model and the subquestion generation function. The function used to map pointers between the two tokenizers did not generalize to the cases here. Examples of these are ['80', '-', 'yard'] and ['80-yard'].

c. 28 samples did not pass through BERTQA successfully, as they exceeded the sequence length (512).

Of the remaining 1189 questions that were processed successfully, we get 55.9% correctly. This still includes questions which are not covered by our subtraction template. We proceed in two ways: First, we filter out questions that MTMSN predicted not to be `addition_subtraction`. This leaves 1106 questions with 59.3% accuracy. The alternative is to filter questions based on their start trigrams, which gives a more relevant set of questions. Of the 1189 questions, 892 start with the phrases 'How many more', 'How many fewer', and 'How many less'. Our model answers 64% of these correctly.

[3] We build upon the implementation of Min et al. (2019)