Improving Numerical Reasoning Skills in the Modular Approach for Complex Question Answering on Text

Xiao-Yu Guo       Yuan-Fang Li       Gholamreza Haffari
Faculty of Information Technology, Monash University, Melbourne, Australia
{xiaoyu.guo, yuanfang.li, gholamreza.haffari}@monash.edu

Abstract

Numerical reasoning skills are essential for complex question answering (CQA) over text. It requires operations including counting, comparison, addition and subtraction. A successful approach to CQA on text, Neural Module Networks (NMNs), follows the programmer-interpreter paradigm and leverages specialised modules to perform compositional reasoning. However, the NMNs framework does not consider the relationship between numbers and entities in both questions and paragraphs. We propose effective techniques to improve NMNs’ numerical reasoning capabilities by making the interpreter question-aware and capturing the relationship between entities and numbers. On the same subset of the DROP dataset for CQA on text, experimental results show that our additions outperform the original NMNs by 3.0 points for the overall F1 score.

1 Introduction

Complex Question Answering (CQA) is a challenging task, requiring a model to perform compositional and numerical reasoning. Originally proposed for the visual question answering (VQA) task, Neural Module Networks (NMNs) (Andreas et al., 2016) have recently been adopted to tackle the CQA problem over text (Gupta et al., 2020). The NMNs is an end-to-end differentiable model in the programmer-interpreter paradigm. Briefly, the programmer learns to map each question into a program, i.e. a sequence of neural modules, and the interpreter then “executes” the program, operationalized by modules, on the paragraph to yield the answer for different types of complex questions. NMNs achieves the best performance on a subset of the challenging DROP dataset (Dua et al., 2019) and is interpretable by nature.

However, NMNs’ performance advantage is not consistent, as it underperforms in some types of questions that require numerical reasoning. For instance, for date-compare questions, MTMSN (Hu et al., 2019) achieves an F1 score of 85.2₁, whereas NMNs’ performance is 82.6. Similarly, for count questions, the F1 score is 61.6 for MTMSN and 55.7 for NMNs. This performance gap stems from two deficiencies of NMNs, which we describe below with the help of two examples in Figure 1.

Firstly, NMNs’ interpreter is oblivious to the question when executing number-related modules. For executing number-related modules, the interpreter only receives the paragraph as input, but not the question. Such a lack of direct interactions with the question impairs model performance: the entities in the question, which may also occur in the paragraph, can help locate significant and relevant numbers to produce the final answer. In the first example in Figure 1, if the interpreter is aware of the correct event mentioned in the question (i.e. “the Constituent Assembly being elected”), it can easily find the same event in the paragraph and further locate its date (“12 November”) precisely. Without this knowledge, the original NMNs found the wrong event (i.e. “dissolved the Constituent Assembly”), thus the wrong date (“January 1918”), leading to an incorrect answer.

Secondly, NMNs disregards the relative positioning of entities and their related numbers in the paragraph. Although NMNs can learn separate distributions over numbers extracted from a paragraph, it does not have an effective mechanism to identify the number that connects to a given entity. Such an ability to recognise the association among numbers and entities is vital for learning numerical reasoning skills: the operation between numbers is meaningful only when they refer to the same entity or the same type of entities. The second example in Figure 1 illustrates the positioning of entities and their related numbers. With only a constraint on a window around an entity, the NMNs’ interpreter tends to identify the nearest number as the related one to a given entity (“August 1996 to December 1997” for entity “PUK and KDP later co-operated”), resulting in wrong predictions.

We propose three simple and effective mechanisms to improve NMNs’ numerical reasoning capabilities.

\[^{1}\text{All F1 and EM numbers in this paper are percentages.}\]
Firstly, we improve the interpreter to make it question-aware. By explicitly conditioning the execution on the question, the interpreter can exploit the information contained in the question. Secondly, we propose an intuitive constraint to better relate numbers and their corresponding entities in the paragraph. Finally, we strengthen the auxiliary loss to increase attention values of entities in closer vicinity within a sentence. Experimental results show that our modifications significantly improve NMNs’ numerical reasoning performance by up to 3.0 absolute F1 points. With minor modification, these mechanisms are simple enough to be applied to other modular approaches.

2 Related Work

Complex Question Answering focuses on questions that require capabilities beyond multi-hop reasoning. These capabilities include numerical, logical and discrete reasoning. A number of neural models were recently proposed to address the CQA task, such as BiDAF (Seo et al., 2017), QANet (Yu et al., 2018), NMNs (Gupta et al., 2020) and NumNet (Ran et al., 2019), which achieved high performance on benchmark datasets such as DROP (Dua et al., 2019).

Numerical Reasoning is an essential capability for the CQA task, which is a challenging problem since the numbers and computation procedures are separately extracted and generated from raw text. Dua et al. (2019) modified the output layer of QANet (Yu et al., 2018) and proposed a number-aware model NAQANet that can deal with numerical questions for which the answer cannot be directly extracted from the paragraph. In addition to NAQANet, NumNet (Ran et al., 2019) leveraged Graph Neural Network (GNN) to design a number-aware deep learning model. Also leveraging GNN, Chen et al. (2020a) distinguished number types more precisely by adding the connection with entities and obtained better performance. Chen et al. (2020b) searched possible programs exhaustively based on answer numbers and employed these programs as weak supervision to train the whole model. Using dependency parsing of questions, Saha et al. (2021) focused on the numerical part and obtained excellent results on different kinds of numerical reasoning questions.

Neural Module Networks (NMNs) (Gupta et al., 2020) adopts the programmer-interpreter paradigm and is a fully end-to-end differentiable model, in which the programmer (responsible for composing programs) and the interpreter (responsible for soft execution) are jointly learned. Specialised modules, such as \texttt{find} and \texttt{find-num}, are predefined to perform different types of reasoning over text and numbers. Compared with those techniques that employ GNNs (Ran et al., 2019; Yu et al., 2018), NMNs is highly interpretable while achieving competitive performance. More details can be found in Appendix A.

3 Proposed Model

In this section, we will discuss the deficiencies of NMNs described in Section 1 and propose three techniques to overcome these problems. Considering the importance of questions while executing programs, we incorporate a question-to-paragraph alignment matrix to form a question-aware interpreter in Section 3.1. In Section 3.2, the correspondence between numbers and their related entities is enhanced with a simple and effective constraint on number-related modules. In Section 3.3, we strengthen the auxiliary loss function in NMNs to further concentrate attention
3.1 Question-aware Interpreter

The interpreter in the NMNs framework is responsible for executing specialised modules given the context (i.e. paragraph). For number-related modules such as “find-num”, the question is not taken into account, which limits NMNs’ performance on numerical reasoning, as information in the question is not taken into account. As an example, let us take a clear look at the “find-num” module in NMNs. 

\[ \text{find-num}(P) \rightarrow T^2. \]

This module takes as input the distribution over paragraph tokens, and produces output an distribution over the numbers:

\[
\begin{align*}
S_{ij}^n &= P_i^T W_n P_{nj}, \quad (1) \\
A_{i}^n &= \text{softmax}(S_{ij}^n), \quad (2) \\
T &= \sum_i P_i A_{i}^n, \quad (3)
\end{align*}
\]

where input \( P \) and output \( T \) are distributions over paragraph tokens and numbers respectively, \( P \) is the paragraph token representations, \( i \) is the index of the \( i^{th} \) paragraph token, \( n_j \) is the index of the \( j^{th} \) number token, and \( W_n \) is a learnable matrix. Note that when computing the similarity matrix between the paragraph token \( P_i \) and the number token \( P_{nj} \) in Equation 1, there is no interaction with the question.

When the correct number types or related entities can be easily found in the question, incorporating the question in “find-num” can help narrow down the search of numbers in the paragraph. The first example in Figure 1 shows that the NMNs fails to locate the correct number as the wrong event is recognized, without interacting with the question.

Inspired by this idea, we propose the question-to-paragraph alignment modification to number-related modules. Specifically, the definition of “find-num” is modified as follows:

\[ \text{find-num}(P, Q) \rightarrow T^n, \]

where the additional input \( Q \) obtained from the programmer represents the distribution over question tokens, and the new output is represented by \( T^n \). Additional computational steps (Equation 4 to 7 below) are added after Equation 3:

\[
\begin{align*}
S_{kj}^{n'} &= Q_k^T W_n P_{nj}, \quad (4) \\
A_{k}^{n'} &= \text{softmax}(S_{kj}^{n'}), \quad (5) \\
T' &= \sum_k Q_k A_{k}^{n'}, \quad (6) \\
T^n &= \lambda \cdot T + (1 - \lambda) \cdot T', \quad (7)
\end{align*}
\]

where \( Q \) is the question token representations and \( k \) is the index of the \( k^{th} \) question token.

As can be seen from the above equations, the input of the improved “find-num” module is extended to include not only paragraph but also question token distributions instead of only the paragraph. More precisely, \( T' \) is another alignment matrix between all question tokens and number tokens, using the same form of Bi-linear attention computation as \( T \).

Finally, the new distribution \( T^n \) is produced by the weighted sum of \( T \) and \( T' \) with an additional hyper-parameters \( \lambda \). Here we fix \( \lambda = 0.5 \) so that NMNs treats the paragraph and the question equally. Other number-related modules are also revised in a similar way, e.g. “find-date”, “compare-num-lt-than”, “find-max-num”.

3.2 Number-Entity Positional Constraint

It is highly likely for a paragraph to contain multiple numbers and entities, as shown in Figure 1. For such paragraphs, the original NMNs allows all numbers to interact with all entities in the computation of number-related modules such as “find-num”. This is detrimental to performance as, intuitively, a number far away from an entity is less likely to be related to the entity. As the second example in Figure 1 shows, NMNs connects “December 1997” to the entity “PUK and KDP” since “2003” is far away from it, resulting in wrong predictions eventually.

To tackle this issue, we add another computational component, the relation matrix \( U^n \), into number-related modules. Taking the “find-num” module as an example, the following step is added before Equation 2 when computing \( S_{ij}^n \):

\[
S_{ij}^n = U_{ij}^n \circ S_{ij}^n, \quad (8)
\]

where \( \circ \) is element-wise multiplication. In the above equation, the value of \( S_{ij}^n \) is updated with the relation matrix \( U^n \), which constrains the relationship between the \( i^{th} \) paragraph token and \( j^{th} \) number token. More specifically, let \( s_t \) be the token index set for the \( t^{th} \) sentence in the paragraph. Thus, if both the \( i^{th} \) paragraph token and the \( j^{th} \) number token belong to

---

2We follow Gupta et al. (2020) and use same variables, annotations in equations for consistency.
We report F1 and Exact Match (EM) scores following the literature (Dua et al., 2019; Gupta et al., 2020).

4 Experiments

4.1 Dataset and Settings

We evaluate model performance on the same subset of the DROP dataset used by the original NMNs (Gupta et al., 2020), which contains approx. 19,500 QA pairs for training, 440 for validation and 1,700 for testing. The training procedures and hyper-parameter settings are the same as the original NMNs (Gupta et al., 2020). We report F1 and Exact Match (EM) scores following the literature (Dua et al., 2019; Gupta et al., 2020).

4.2 Results

Table 1 shows the main results, where “original” represents the performance of the original NMNs (Gupta et al., 2020). Row 4, “+qai+nepc+aux”, is our full model, which includes the question-aware interpreter (+qai), the number-entity positional constraint (+nepc), and the improved auxiliary loss (+aux). It can be observed that compared to “original”, our full model achieves significantly higher performance with F1 of 80.4 and EM of 76.6, representing an increase of 3.0 and 2.6 absolute points respectively. Besides, our significant test shows $p \leq 0.01$.

Table 1: Comparison between different models.

| Methods                          | F1  | EM  |
|----------------------------------|-----|-----|
| original(Gupta et al., 2020)     | 77.4| 74.0|
| ours                            |     |     |
| +qai                            | 79.0| 74.9|
| +nepc                           | 79.9| 76.0|
| +aux                            | 80.4| 76.6|

Table 2: Performance (F1) by question types.

| Question Type     | MTMSN | original | ours |
|-------------------|-------|----------|------|
| date-compare      | 85.2  | 82.6     | 86.0 |
| date-difference   | 72.5  | 75.4     | 78.6 |
| number-compare    | 85.1  | 92.7     | 90.1 |
| extract-number    | 80.7  | 86.1     | 90.1 |
| count             | 61.6  | 55.7     | 61.8 |
| extract-argument  | 66.6  | 69.7     | 73.2 |

We also conduct an ablation study to discuss the contribution of individual technique. The second line, “+qai”, is the results with the question-aware interpreter employed only. For this variant, the F1 and EM scores improve on the original baseline by 1.6 and 0.9 points respectively. With the addition of the number-entity positional constraint, “+nepc”, results show an improvement of 2.5 and 2.0 points for F1 and EM when comparing with “original”. These results show that all of the three techniques are effective in improving numerical reasoning skills for NMNs.

We also report performance by subsets of different question types in Table 2. Except for the number-compare type, our model improves on the original NMNs across all other types of questions significantly, by at least 3.2 absolute points for F1. In addition, our model outperforms aforementioned MTMSN (Hu et al., 2019) on all question types as well.
5 Conclusion

Neural Module Networks (NMNs) represent an interpretable state-of-the-art approach to complex question answering over text. In this paper, we further improve NMNs’ numerical reasoning capabilities, by making the interpreter question-aware and placing stronger constraints on the relative positioning of entities and their related numbers. Experimental results show that our approach significantly improves NMNs’ numerical reasoning ability, with an increase in F1 of 3.0 absolute points.

References

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In Proceedings of CVPR, pages 39–48.

Kunlong Chen, Weidi Xu, Xingyi Cheng, Zou Xiaochuan, Yuyu Zhang, Le Song, Taifeng Wang, Yuan Qi, and Wei Chu. 2020a. Question directed graph attention network for numerical reasoning over text. In Proceedings of EMNLP, pages 6759–6768.

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V. Le. 2020b. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In Proceedings of ICLR.

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 Proceedings of HLT-NAACL, pages 2368–2378.

Nitish Gupta, Kevin Lin, Dan Roth, Sameer Singh, and Matt Gardner. 2020. Neural module networks for reasoning over text. In Proceedings of ICLR.

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 EMNLP-IJCNLP, pages 1596–1606.

Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. 2019. NumNet: Machine reading comprehension with numerical reasoning. In Proceedings of EMNLP-IJCNLP, pages 2474–2484.

Amrita Saha, Shafiq R. Joty, and Steven C. H. Hoi. 2021. Weakly supervised neuro-symbolic module networks for numerical reasoning. CoRR, abs/2101.11802. Version 1.

Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2017. Bidirectional attention flow for machine comprehension. In Proceedings of ICLR.

Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. 2018. Qanet: Combining local convolution with global self-attention for reading comprehension. In Proceedings of ICLR.
A NMNs model overview

In order to solve the complex question answering problem, Gupta et al. (2020) proposed a Neural Module Networks (NMNs) model. Consisting of a programmer and an interpreter, NMNs can be more interpretable as shown in Figure 2.

![Figure 2: Architecture of the NMNs model.](image)

As Figure 2 shows, NMNs takes the question and the paragraph as inputs. The programmer firstly maps the question into corresponding “discrete” modules in order. Then, the interpreter executes these generated modules against the corresponding paragraph to produce the final answer. Moreover, all modules are differentiable so that the whole NMNs can be trained in an end-to-end way.

B Settings for Experiments

We mainly use PyTorch and AllenNLP deep learning platforms to implement our model. After 40-epoch training on Ubuntu 16.04 with one V100 GPU Card (16GB memory), it takes around 24 hours to converge. And all reported results are produced based on the saved checkpoint.

| Name          | Value |
|---------------|-------|
| batch size    | 4     |
| epochs        | 40    |
| hard em epochs| 5     |
| learning rate | 1e-5  |
| drop out rate | 0.2   |
| max question length | 50     |
| max paragraph length | 459    |
| max decode step | 14     |

Table 3: Hyper-parameter settings.

For hyper-parameters in our model, we don’t conduct experiments on their search trials since we employ the same settings as Gupta et al. (2020) did, which can be found in Table 3. Note that they are also the configuration to obtain the best performance. For the added parameter $\lambda$ in Equation 7, we leverage an empirical value $\lambda = 0.5$ without any fine-tuning.

Due to the page limitation, we didn’t include more baselines, such as NAQANet (Dua et al., 2019). After running on the same split of DROP dataset, the F1 and EM scores by NAQANet are 62.1% and 57.9% respectively, which are substantially lower than our results in Table 1, by over 17% for both scores. And we did apply these components in Section 3 to other modules, such as the “extract-argument” module (extracts spans or tokens from paragraphs), and also obtained better results (0.5% F1 increase). Besides, for different question types, their statistics on the test set can be found in Table 4.

| Question Type     | Percentage |
|-------------------|------------|
| date-compare      | 18.6%      |
| date-difference   | 17.9%      |
| number-compare    | 19.3%      |
| extract-number    | 13.5%      |
| count             | 17.6%      |
| extract-argument  | 12.8%      |

Table 4: Percentage by question types.

Current NMNs (Gupta et al., 2020) does not support other arithmetic datasets, since some arithmetic operations, including addition, are not supported. Extending related arithmetic modules is one of our future work, based on which the NMNs could be trained on other datasets.