Defining and Testing Essential Number Properties for LMs

Karan Raj Bagri & Dhamale Vivek Shrikrishna & Pawar Rushikesh Gajanansa
Indian Institute of Science
Bengaluru, KA, India
{karanraj,viveksd,rushikeshp}@iisc.ac.in

Abstract

Language Models have achieved impressive feats in Natural Language Processing, excelling at tasks like Text Generation, Machine Translation, Question Answering and Text Summarization. However, NLP systems rarely give special considerations to numbers. They are treated just like any other text tokens, but there is a fundamental difference between words/letters and numbers. Also, during pre-processing most of the numbers get mapped to <UNK> token because of absence from the vocabulary. This results in a relatively poor numerical ability compared to linguistic proficiency. In this project, we come up with a set of properties that LMs should know about numbers, and we build tests to check how well numeracy is captured by current state-of-the-art models. We experiment with different numeral embeddings observing how performance is impacted. Code is available at https://github.com/rushikeshpawar22581/NLP_project_EssentialNumberPropertiesForLMs

1 Introduction

NLP systems neglects numbers. They are treated just like any other word/character tokens even though they are semantically different from texts, and their operations are also different. This affects the ability of models to handle and interact with numbers on numerical tasks such as comparing, sorting, identifying patterns in sequences. Even the large language models, which perform exceedingly well in linguistic tasks, do not show consistency in performance on basic tasks like the ones mentioned above. Although there have been attempts made to represent numbers, that is, employ numerical embeddings to improve the numeracy ability of the models, they do not solve the problem. These are elaborated upon in the next section. We identify properties of numbers that the language models should learn in order to truly understand numbers and perform well on tasks involving numeracy. We develop tests to check if the models are learning these properties. We evaluate the performance of certain state-of-the-art language models on these tests. We test if and how mcuh different numeral embeddings affect the performance of models on these tests.

2 Related Work

Wallace et al. (2019) found that models fail to extrapolate well to numbers outside its training range. Upon making some changes in the validation data, like generating a random number and multiplying the numbers in each paragraph, the performance drop was significant. Authors hypothesised that the source of numeracy are the token embeddings themselves and the reason behind the success or failure of a model on numerical tasks lies in the information encoded within these embeddings. The extrapolation problem persisted for tasks like decoding and addition, while it had minimal effect on the list-maximum finding problem. Pal & Baral (2021) tested T5 models on simple taks in interpolation and extrapolation settings and found the extrapolation performance to be much worse.

Thawani et al. (2021b) have done a survey of the then existing work on numeracy and broken down notion of numeracy into 7 subtasks(tasks to evaluate numeracy like simple arithmetic,

numeration, magnitude comparison, arithmetic word problems, exact facts,measurement estimation, Numerical language modeling).

Testolin (2024) survey shows that state-of-the-art architectures often fall short when probed with relatively simple tasks designed to test basic numerical and arithmetic knowledge. They found that neural networks with external memory show good extrapolation ability. Then, they talk about ad-hoc deep learning architectures.Also using 'scratchpads' and/or 'chain-of-thought' reasoning helps.

Zhang et al. (2020) This paper identified contextual information in pretraining and numeracy as two key factors affecting their performance. Scalar probing : a task of predicting a distribution of possible values for this attribute, and compare it to a ground truth distribution of such values. It shows BERT and ELMo, perform better than non-contextual ones, like Word2Vec, on scalar probing despite the task being non-contextual. Their results imply that scale representation in contextual encoders is mediated by transfer of magnitude information from numbers to nouns in pretraining and making this mechanism more robust could improve performance on this and other CSR tasks.

Ren & Du (2020) consider the magnitude aspect of numeracy for models. Magnitude concerns how well the embedding of numbers encode their values information.Authors propose a new test called SC-k. It is shown that there exists a low dimensional subspace in the original space of number embedding such that a larger degree of magnitude quantified by SC-k is uncovered via projecting the embedding of numbers onto that subspace.

The study by Muffo et al. (2023) compares performance of two GPT2 model fine tuned to perform arithmetic operations on same dataset, one does operations on decomposed numbers and another without number decomposition. Results are observed as model with number decomposition does much better than model without number decomposition. and shows that decomposition during training improves addition and subtraction performance.

Nogueira et al. (2021) concluded that surface form of a number has a strong influence on a model's accuracy. Sub-word representations don't work because two very similar numbers may be represented very differently. Also, extrapolation remains a problem. Saw that representing, for eg, number 32 as "3 10e1 2" scaled best to addition and subtraction on numbers having a large number of digits. Their experiments used T5 model. Thawani et al. (2021a) show that specialized number encoders are helpful in improving the word prediction ability of a language model and gains are also observed on text without numbers. This implies that specialized encoding enhances the numeracy of language models.

Jin et al. (2021) proposed a novel NumGPT model in which they use numeral embeddings for the numbers. Each embeding has 2 parts: one for exponent and another for mantissa. Another study Thawani et al. (2021a) also observed that best performing number encoder is Exp. Spithourakis & Riedel (2018) explores different strategies for modelling numerals, such as memorisation and digit by digit comparison and propose a NN that uses continuous pdf to model numerals from an open vocabulary.

Sundararaman et al. (2020) proposed a new methodology to learn word embedding called DICE (determinant and independent of corpus) for embedding such that their cosine similarity reflects the actual distance on the number line. It also introduced a regularization approach to learn model-based embedding of numbers in a contextual setting. DICE embeddings are learned independently of corpus and effectively capture properties of numeracy. They explicitly associated the embedding of a number in digit form and in word form to have the same embedding.

Jiang et al. (2020) propose to represent the embedding of a numeral as a weighted average of a small set of prototype of number embeddings which can handle out of vocabulary problem for numerals. They squish numbers into log space and use SOMs, GMMs to find prototype numbers, whose embeddings are learnt through any word2vec/Glove.

Peng et al. (2023) leverage numerical properties to enhance the capabilities of the decoder. Authors propose a numeracy-enhanced token embedding method, which fuses the explicit numerical feature with the contextual feature for number tokens, enabling the decoder to perceive numerical properties during the inference. They also propose five nu-

meracy features that can influence the decoding process, that are numerical type feature (int,decimal,fraction,percent, etc), numerical magnitude, numerical unit present, Numerical ratio features, Numerical digit feature.

A study by Kim et al. (2021) tries to solve the problem of extrapolation as observed by Pal & Baral (2021) and Wallace et al. (2019). They Propose a new surface form called 'E digit', which alleviates the problem of extrapolation on DROP (a reading comprehension benchmark Dua et al. (2019)). This uses the digit-position embeddings in addition to digit form. For eg, 2015 is "2 *e* 3 0 *e* 2 1 *e* 1 5 *e* 0" i.e., (digit, e, position).

Feng et al. (2022) pretrain BERT in 2 proposed ways that encourages the models to understand magnitude 10^x and value(in front of 10^x) and encode the dependency between a number and its context.

We notice that in recent times, most of the work in numeracy has been testing the numerical reasoning abilities of the state-of-the-art models on datasets such as Dua et al. (2019) and Mishra et al. (2022). This type of evaluation may be influenced strongly by context. We explore numeracy at a lower level, by evaluating current state-of-the-art models on more basic tasks.

3 Methodology

We conduct tests to understand capture whether models are able to capture following nuances of numbers.

(a) Magnitude: exact value , order of magnitude.

(b) Influence of types of numbers: integer, fraction, decimal, odd/even, positive/negative.

(c) Scaling: in-domain to out of domain performance shift.

We perform simple addition of two integers (ranging from 1 digit to 60 digits in length) to check how scaling influences the model's performance. We ensure representation of positive and negative numbers in this dataset to check if there are performance differences for numbers of the two signs.

We design tests to investigate the numerical comparison capabilities of the models. We use list min-max and list sort to evaluate this. For list min-max and for list sort, we generate lists of different lengths, containing numbers of different magnitudes and different signs. This helps us check how much the differing magnitudes and signs affect the comparison capability. Experimental setups are slightly different for different models. We elaborate upon them in the subsequent sections.

We run these tests on GPT-3.5 Turbo and LLama 2 quantized. We access GPT-3.5 Turbo through the OpenAI API. We evaluate the quantized version of LLama 2 by locally hosting it. Since this takes time, only a fraction of the integer addition dataset could be used for evaluation.

4 Results

4.1 Evaluation of Llama 2 (Quantized):

Llama 2 is a family of pre-trained and fine-tuned large language models (LLMs) released by Meta AI in 2023 In this implementation, we use llama.cpp. llama.cpp's objective is to run the LLaMA model with 4-bit integer quantization. GGML, a C library for machine learning, facilitates the distribution of large language models (LLMs). It utilizes quantization to enable efficient LLM execution on consumer hardware. The Hugging Face community provides many quantized models. We use 'TheBloke/Llama-2-13B-chat-GGML.

4.1.1 Investigating Integer addition ability:

1. We use text generation(completion) to get the answer from llama model. As explained before, synthetically generated dataset of addition is used to carry out this test.

2. We test nearly 952 examples from our addition dataset. After getting results, values which are not numbers such examples are ignored.

3. Since we run the model locally, sequentially processing of single prompt takes nearly 7 to 7.5 sec. Hyper parameters used: max tokens=256, temperature=0.5, top p=0.95,repeat penalty=1.2, top k=150 (referred from hugging face page).

4. results: These results show that quantization of model severely affects performance of model of mathematical operations.

5. When questions are in format (+ve , -ve) -> true value sign
   Number of wrong predictions when (+ve , -ve) -> +ve : 16/112
   Number of total wrong when (+ve, -ve) : 128/250

6. When questions are in format (-ve , +ve) -> true value sign
   Number of wrong predictions when (-ve , +ve) -> -ve : 74/90
   Number of total wrong when (-ve, +ve) : 153/199

No of Digits	Number of Data Points	Wrong Predictions	Percentage Wrong Predictions
1	200	43	21.5
2	400	157	39.25
3	349	146	41.83381089

Data Points Type	Number of Data Points	Wrong Predictions	Percentage Wrong Predictions
Positive	300	16	5.333333333
Negative	649	330	50.84745763
Both Negative	200	49	24.5
Only One Negative	449	281	62.58351893
Total	949	346	36.45943098
sign mistakes	330	44	13.33333333

Table 1: Llama addition results

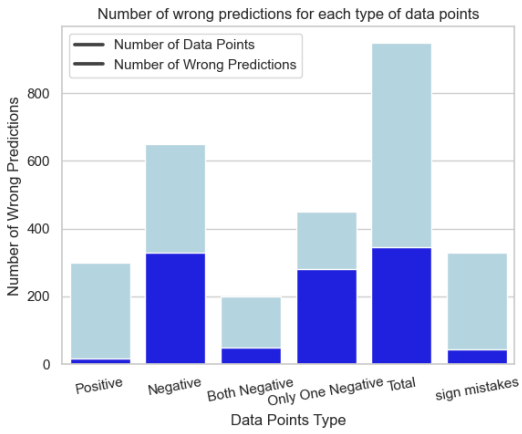


Figure 1: Llama addition plot

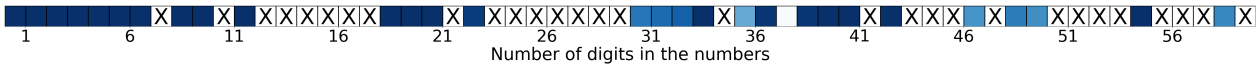
4.2 Evaluation of GPT-3.5 Turbo:

In this section, we evaluate the integer addition and integer comparison capabilities of GPT-3.5 Turbo. We access this model by making API calls to the OpenAI server.

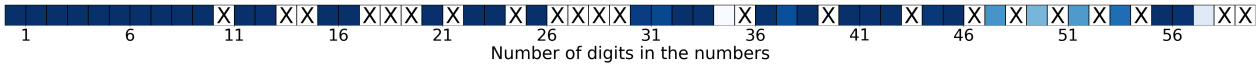
4.2.1 Investigating integer addition ability:

We attempt to quantify the ability of this model to add 2 integers. We generate a balanced dataset (having 24000 samples consisting of 3-element lists of $[a, b, a+b]$, taking inspiration from the addition task and balanced sampling mentioned by Nogueira et al. (2021). We vary the number of digits from 1 to 60. To check whether a number being positive or negative influences the accuracy of the predictions, we make sure that we have equal representation of all 4 combinations of signs of a and b {(+ve, +ve), (+ve, -ve), (-ve, +ve), (-ve, -ve)}. We specify our request and response formats to the model via the prompts. Our requests have the format "What is {a} + {b}?". We expect responses in the format "{a} + {b} = {c}", where c is the predicted sum. We batch requests together to comply with the rate limits of API access, by concatenating multiple requests into a single request.

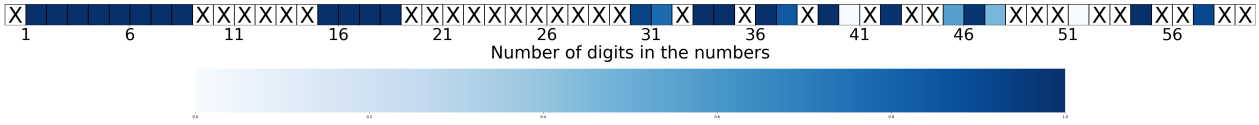
The model does not always respond in the correct format. Moreover, sometimes the model does not respond to all the requests in a single batched request. We manually figure out all the response formats. We consider only those responses that contained $\{a\}$, $\{b\}$ and $\{c\}$. There happened to be 6 such formats, which were then processed to reconstruct the triplets (a, b, c). The results obtained are shown in figure 2.



(a) Accuracy of addition when both integers are positive



(b) Accuracy of addition when at least one of the integers is negative



(c) Accuracy of addition when both integers are negative

Figure 2: Integer Addition Accuracy Heatmaps

Note that the crossed out cells represent the datapoints for which we had less than 40 sample triplets (a, b, c). We can see that with the increase in number of digits, the accuracy drops. This is consistent with what Nogueira et al. (2021) had observed. As the model encountered smaller numbers more during its training especially in the context of numerical arithmetic operations, it tends to perform well when adding two numbers comprising a relatively low number of digits. It does not do well when presented with numbers that did not occur or occurred less frequently in its training data Kim et al. (2021). Sometimes, it returned garbage values like *"13080003539169249962Andgen9ynomial9257"* or *"Editorial58991799307800953 60"*, heavily indicative of its bias towards alphanumeric words in the training corpus. Also, the drop in accuracy is not monotonic, which could be

down to the random nature of the model's behaviour. This makes it difficult to compare the three heatmaps to conclude if the model behaves less or more accurately when dealing with negative numbers. We would like to sample more but are limited by the rate limits on access of the API.

4.2.2 Investigating the ability to compare numbers:

To evaluate this, we consider two tasks, finding the maximum and minimum elements in a list and sorting a list, taking inspiration from Wallace et al. (2019) and Pal & Baral (2021).

For list min-max, we generate lists of length 3, lists of length 5 and lists of length 10, containing numbers having upto d digits, where d is varied from 1 to 5. We ensure that half the lists are such that all the numbers are positive and half the lists may contain negative numbers (probability of a number being negative is set to 0.5). Our requests for finding the list minimum have the format "Find the minimum number in the list {list}". We expect responses in the format "Min({list}) = {min num}". Again, we batch multiple requests into one. Similar formats are used for finding the maximum. The results are shown in figure 3.

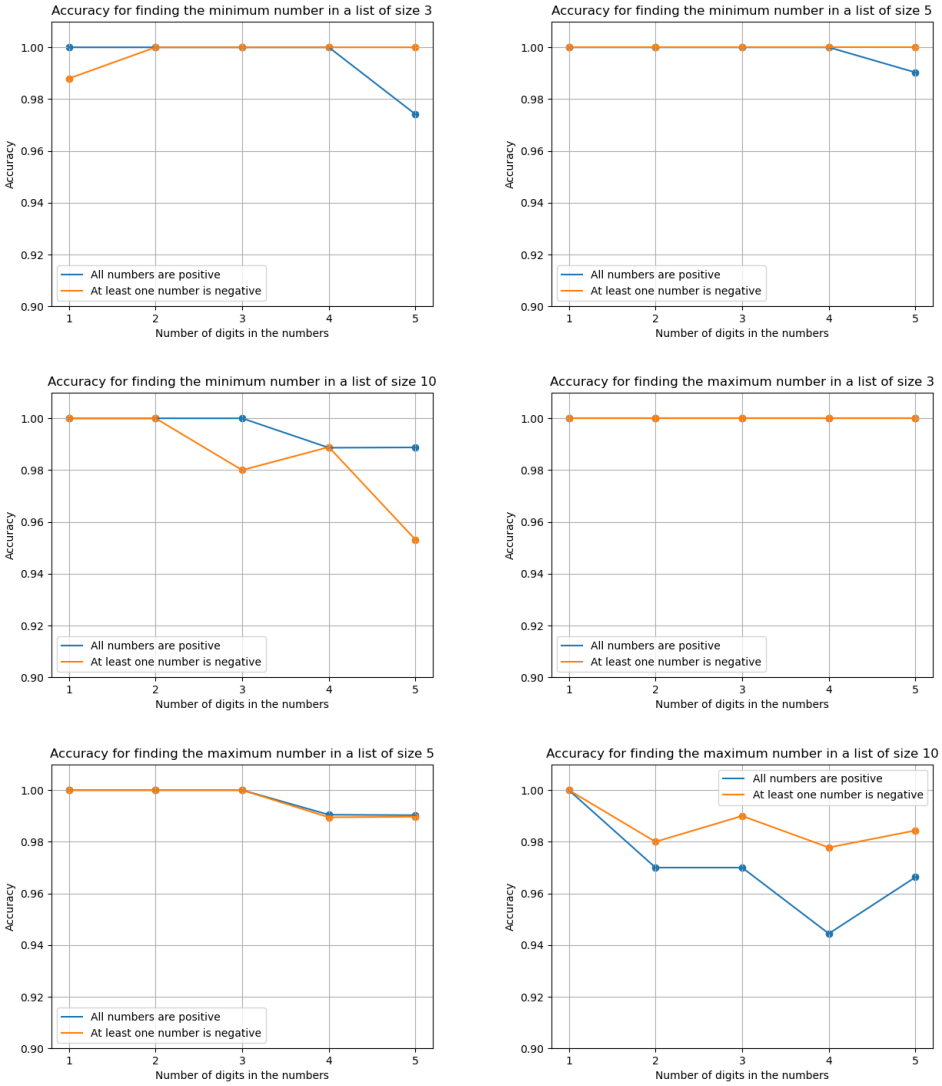


Figure 3: Evaluating list min-max accuracy of GPT-3.5 turbo

We see that as the magnitude range (number of digits) of the number increases, the accuracy goes down. Here, even though the numbers encountered are well within the model's training range, it seems to experience some difficulty for the larger numbers. One can imagine the accuracy being much worse when the magnitude range is much bigger than the ones we have tested for. Also, as the size of the list increases, the accuracy falls. One more important observation is that when the model is finding the minimum element of a list containing 10 elements, the accuracy for cases where all the numbers are positive is slightly higher. Conversely, when the model is looking for the maximum element, the presence of negative numbers increases the accuracy. This could suggest that the model understands how to separate positive and negative numbers better than how it understands how to differentiate between numbers only on the basis of magnitude.

For list sort, we use the same lists we had generated for list min-max. Our requests are of the format "Sort the numbers in the list {list}". We expect the response to be in the format "{Sorted list}". The results are shown in figure 4.

Again, we see that the accuracy clearly drops. As the magnitude range of the number increases, there is a slight drop. But as the size of the list increases, there is an even more significant drop. Thus, confirming the fact that the model, at least in zero-shot or one-shot (since we are giving it an example to show the expected response format) settings, is unable to do comparisons between many numbers.

5 Conclusion

We saw that the quantized version of Llama-2 did not perform well on the integer addition task. It exhibited a high error rate even for small numbers. In addition, it performed much worse in cases where one of the numbers was negative. GPT-3.5 Turbo performed well on the integer addition task. However, as the number of digits increased, the accuracy went downwards. Also, the model became more likely to give garbage responses. All this suggests that the model does not truly understand the addition task. But the signs of the numbers did not influence the addition accuracy. We also saw that although the list min-max performance reduced with increasing size of list and magnitude of numbers, it was still fairly good; suggesting that the model is good at differentiating one number from the rest. An interesting observation was that the presence of negative numbers actually helped with the list maximum task. This suggests that the model was using the sign information well. For list-sort however, the performance drop was, expectedly, more significant since more comparisons had to be performed.

In the future, we plan to look for a way to access Llama-2 (not quantized) through API calls to have batch processing of prompts to see actual results. We plan to add more tests to capture the numeracy ability of models in more details. We plan to try out different numeral embeddings to check how they influence the models' performance on these tests.

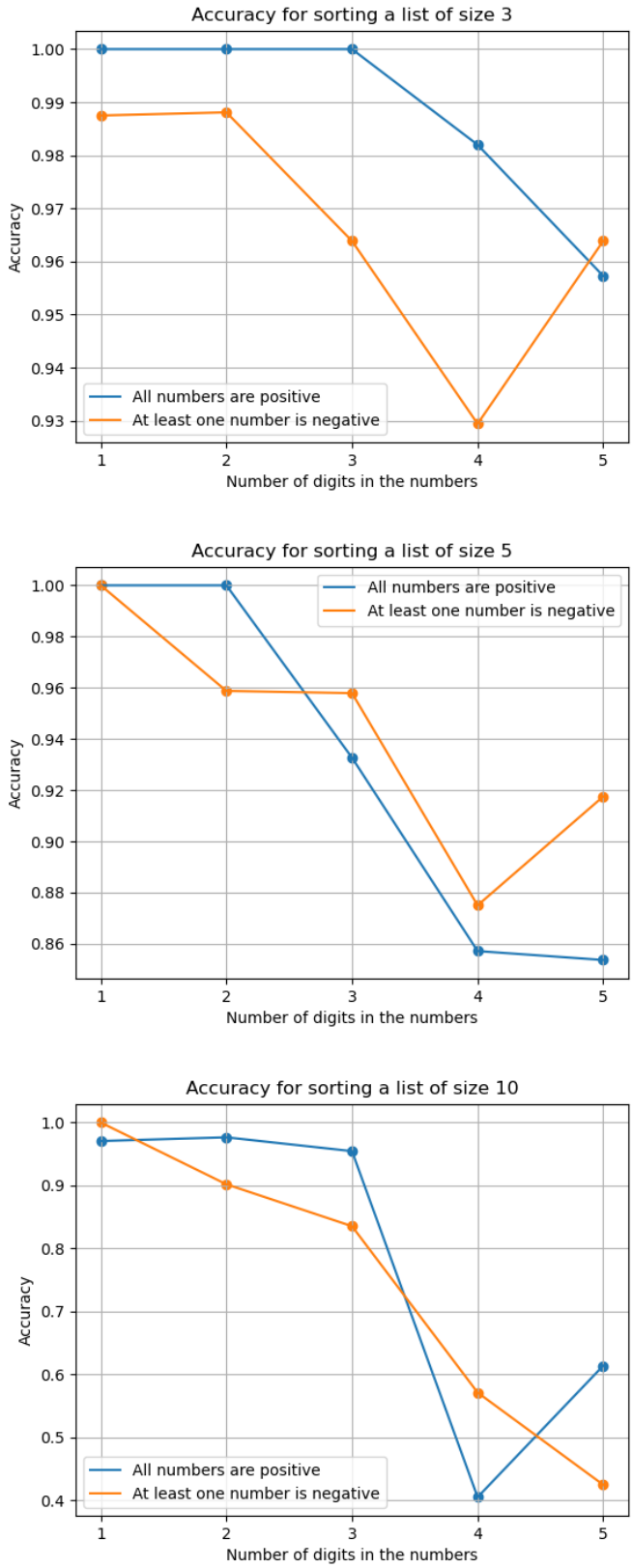


Figure 4: Evaluating list sorting accuracy of GPT-3.5 Turbo

Contributions

Pawar Rushikesh Gajanansa:

Literature Review:(9 papers)

Thawani et al. (2021b),Peng et al. (2023),Muffo et al. (2023),Thawani et al. (2021a),Feng et al. (2022),Naik et al. (2019), Jiang et al. (2020),Spithourakis & Riedel (2018),Mishra et al. (2020)

Implementation: Compiled literature review.
Note: Rushikesh couldn't contribute more due to an emergency medical situation. He is still having to stay under observation. He intends to compensate in end term work.

Karan Raj Bagri:

Literature Review:(7 Papers)

Wallace et al. (2019), Jin et al. (2021), Kim et al. (2021), Testolin (2024), Nogueira et al. (2021), Dua et al. (2019), Mishra et al. (2022)

Implementation:
Preparation of dataset and Evaluation of GPT-3.5 Turbo.

Dhamale Vivek Shrikrishna:

Literature Review:(8 Papers)

Geva et al. (2020),Ren & Du (2020),Al-Negheimish et al. (2021),Andor et al. (2019),Geva et al. (2020),Pal & Baral (2021),Ran et al. (2019),Sundararaman et al. (2020)

Implementation:
Evaluation of Llama 2 (Quantized)

References

Hadeel Al-Negheimish, Pranava Madhyastha, and Alessandra Russo. Numerical reasoning in machine reading comprehension tasks: are we there yet?, 2021.

Daniel Andor, Luheng He, Kenton Lee, and Emily Pitler. Giving bert a calculator: Finding operations and arguments with reading comprehension, 2019.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs, 2019.

Fuli Feng, Xilin Rui, Wenjie Wang, Yixin Cao, and Tat-Seng Chua. Pre-training and evaluation of numeracy-oriented language model. In Proceedings of the Second ACM International Conference on AI in Finance, ICAIF '21, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391481. doi: 10.1145/3490354.3494412. URL https://doi.org/10.1145/3490354.3494412.

Mor Geva, Ankit Gupta, and Jonathan Berant. Injecting numerical reasoning skills into language models, 2020.

Chengyue Jiang, Zhonglin Nian, Kaihao Guo, Shanbo Chu, Yinggong Zhao, Libin Shen, and Kewei Tu. Learning numeral embeddings, 2020.

Zhihua Jin, Xin Jiang, Xingbo Wang, Qun Liu, Yong Wang, Xiaozhe Ren, and Huamin Qu. Numgpt: Improving numeracy ability of generative pre-trained models, 2021.

Jeonghwan Kim, Giwon Hong, Kyung-min Kim, Junmo Kang, and Sung-Hyon Myaeng. Have you seen that number? investigating extrapolation in question answering models. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.),

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 7031–7037, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.563. URL https://aclanthology.org/2021.emnlp-main.563.

Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Sachdeva, and Chitta Baral. Towards question format independent numerical reasoning: A set of prerequisite tasks, 2020.

Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Sachdeva, Peter Clark, Chitta Baral, and Ashwin Kalyan. Numglue: A suite of fundamental yet challenging mathematical reasoning tasks, 2022.

Matteo Muffo, Aldo Cocco, and Enrico Bertino. Evaluating transformer language models on arithmetic operations using number decomposition, 2023.

Aakanksha Naik, Abhilasha Ravichander, Carolyn Rose, and Eduard Hovy. Exploring numeracy in word embeddings. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 3374–3380, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1329. URL https://aclanthology.org/P19-1329.

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks, 2021.

Kuntal Kumar Pal and Chitta Baral. Investigating numeracy learning ability of a text-to-text transfer model, 2021.

Rao Peng, Chuanzhi Yang, Litian Huang, Xiaopan Lyu, Hao Meng, and Xinguo Yu. A numeracy-enhanced decoding fornbsp;solving math word problem. In Natural Language Processing and Chinese Computing: 12th National CCF Conference, NLPCC 2023, Foshan, China, October 12–15, 2023, Proceedings, Part III, pp. 111–122, Berlin, Heidelberg, 2023. Springer-Verlag. ISBN 978-3-031-44698-6. doi: 10.1007/978-3-031-44699-3_11. URL https://doi.org/10.1007/978-3-031-44699-3_11.

Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. Numnet: Machine reading comprehension with numerical reasoning, 2019.

Yuanhang Ren and Ye Du. Enhancing the numeracy of word embeddings: A linear algebraic perspective. In Xiaodan Zhu, Min Zhang, Yu Hong, and Ruifang He (eds.), Natural Language Processing and Chinese Computing, pp. 170–178, Cham, 2020. Springer International Publishing. ISBN 978-3-030-60450-9.

Georgios Spithourakis and Sebastian Riedel. Numeracy for language models: Evaluating and improving their ability to predict numbers. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, 2018. doi: 10.18653/v1/p18-1196. URL http://dx.doi.org/10.18653/v1/P18-1196.

Dhanasekar Sundararaman, Shijing Si, Vivek Subramanian, Guoyin Wang, Devamanyu Hazarika, and Lawrence Carin. Methods for numeracy-preserving word embeddings. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 4742–4753, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.384. URL https://aclanthology.org/2020.emnlp-main.384.

Alberto Testolin. Can neural networks do arithmetic? a survey on the elementary numerical skills of state-of-the-art deep learning models. Applied Sciences, 14(2):744, January 2024. ISSN 2076-3417. doi: 10.3390/app14020744. URL http://dx.doi.org/10.3390/app14020744.

Avijit Thawani, Jay Pujara, and Filip Ilievski. Numeracy enhances the literacy of language models. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 6960–6967, Online and Punta Cana, Dominican Republic, November 2021a. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.557. URL https://aclanthology.org/2021.emnlp-main.557.

Avijit Thawani, Jay Pujara, Pedro A. Szekely, and Filip Ilievski. Representing numbers in nlp: a survey and a vision, 2021b.

Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. Do nlp models know numbers? probing numeracy in embeddings, 2019.

Xikun Zhang, Deepak Ramachandran, Ian Tenney, Yanai Elazar, and Dan Roth. Do language embeddings capture scales?, 2020.