Transformer Models with Contextualized Late Interaction for Math Information Retrieval

David Na
Johns Hopkins University
dna5@jhu.edu

Abstract

Math Information Retrieval (MIR) has had many major advancements following the two major Math IR workshops, NTCIR and ARQMath. To this day, the most effective systems are built from classic retrieval methods that involve indexing text and building formula vectors. However, recent studies have shown the increasing performance of bi-encoder models in MIR. Submissions to ARQMath-3 have shown Bi-Encoder models can significantly improve performance. This study attempts to build on existing MIR transformer model architectures by introducing a *late interaction* step first introduced by *ColBERT - Contextualized Late Interaction over BERT -* (Khattab and Zaharia, 2020; Santhanam et al., 2021). This late interaction involves a pre-computation step where document embeddings are pre-computed and stored on disk. Using these pre-computed embeddings, the model can compute document similarity scores and retrieve documents at an accelerated pace. I evaluate several existing pre-trained BERT models with this contextualized late interaction. In summary, my evaluations results found that applying contextualized late interaction significantly improves runtime performance at the cost of classification accuracy. Results still significantly outperform baseline models but are not as robustly accurate as the current transformer models.

1. Introduction

Math Information Retrieval is a niche application in the greater study of Information Retrieval. It involves the development of "math-aware" search engines that can retrieve relevant information from documents with math formulas and concepts embedded in them. A simple example to help contextualize MIR: Given a math formula, retrieve all documents containing the same or remarkably similar formula from Wikipedia. Additionally, the number of document collections that contain Math-related information has grown substantially and with room to continue to grow at an accelerated rate. Classic IR systems do not account for indexing math formulae. It is not uncommon practice to remove such text via tokenization and stemming. This is problematic as the benefits of Math IR is intuitive. As a student in computer science, math formulae are in every one of my class resources. It would be quite beneficial to be able to search for answers quickly and accurately from such a collection of documents. Imagine being able to retrieve information from all lecture notes with a single query. The potential benefit for researchers, students and academics is limited only by one's own imagination.

ARQMath and NTCIR 10-12 are the main workshops that were focused on MIR. NTCIR 10, 11 and 12 were held from 2014-2016 and run by the National Institute of Informatics of Tokyo, Japan. This is the first shared task workshop geared toward Math Information Retrieval.

The NTCIR workshop had three main tasks. Formula search which involved finding similar formulae for a given formula query. Formula and Text search which involved searching documents in a collection for a combination of keywords and formulas. And finally, open information retrieval, which involved searching the math collection using full text queries. NTCIR used data from a Wikipedia dataset of Math tagged articles and an arXiv dataset consisting of scientific papers. ARQMath was held from 2019-2021 and hosted by the Rochester Institute of Technology. The main dataset for ARQMath were Math Stack Exchange posts taken from 2010 to 2021 with the most recent year being the testing set and prior years being the validation set. ARQMath similarly had three tasks. Task one was Answer retrieval. Given a topic (a math question posted on stack exchange), return answers that relevantly address that topic. Task two was Formula Retrieval. Given a formula, retrieve documents that contain that formula or a similar formula. And task 3 – introduced in ARQMath 3 – which involved Open Domain Question Answering where you can retrieve system answers from any dataset source.

For my study, I focused on Task 1 introduced by the ARQMath 3 workshop. I used the Math Stack Exchange collection to train my models and the trec-eval tool for scoring my models relative to the list of relevant documents per query. I evaluated three different model architectures with five different model types. I evaluated a BERT based model called MathBERT (Peng et al, 2021), which was pretrained to recognize math formulae. I evaluated two Albert (A Lite BERT) model from a submission team in ARQMath-3 (Reusch et al, 2022). They provide two versions of Albert. One that was fully tuned and trained for math IR and a pretrained model that had been trained on formula understanding. I also use a Bert base model that they pretrained in the same way they pretrained their Albert model. Finally, I modified a MPNet model and fully trained this model on the Math Stack Exchange data. I took the base MPNet-v2 model (Song et. Al, 2020)

2. Related Work

There have been many studies that have gone into how to create a MIR system that is capable of understanding math formulae. Most of these studies have resulted from the NTCIR and ARQMath conferences. First, I will talk about the most important studies covering the basics of Math IR. The most prominent contribution to Math IR is the development of efficient tokenization methods for math formulae. There are various structures developed to be able to capture additional information from standard math formula representations, such as LaTeX and MathML. Math IR introduced two new concepts namely the Symbol Layout Tree (SLT) and the Operator Tree (OPT), which I will discuss in the next section.

2.1 Math Representations

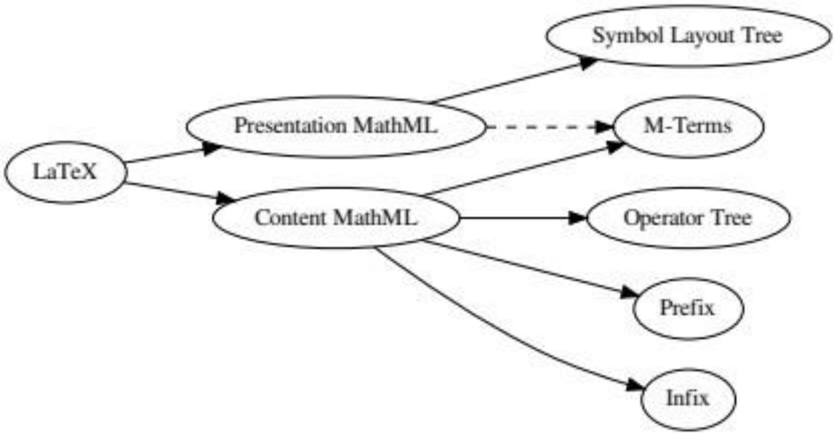
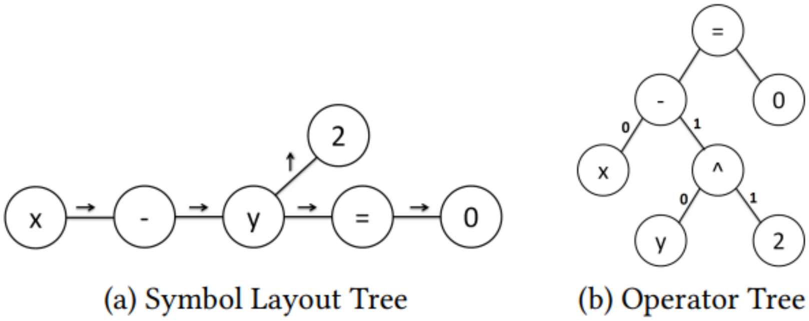


Figure 1: Tree of Math Representations[1]

The figure above shows the general progression of math representations (Novotny et al, 2020). LaTeX is the standard math representation used to write formulas to documents such as PDFs. Novotny explains that the problem with LaTeX is that it is a Turing-complete language meaning that it is statistically impossible to parse generally. From LaTeX we have another representation called MathML where there are two presentation formats: Presentation MathML (PMML) and Content MathML (CMML). The difference between these two is that PMML captures operations as they are presented, while CMML captures correct order of operations. Functions such as double factorial are encoded as “!!” while in CMML all operations are tokenized, so “!!” will be encoded as a function call denoting a double factorial *(<csymbol cd= “latexml”>double-factorial</csymbol>*).

2.1b Tree Representations



(a) Symbol Layout Tree

(b) Operator Tree

Figure 2: Tree representations[2] for $x - y^2 = 0$

[1] https://ceur-ws.org/Vol-2696/paper_235.pdf

[2] https://dl.acm.org/doi/pdf/10.1145/3077136.3080748

Figure 1 shows the tree representations for $x - y^2 = 0$. Following ARQMath and the NTCIR conferences, these trees became the standard for encoding Math formulae as they were much easier to index without loss of information. These symbol tree representations were introduced by a math IR system called Tangent-S (Davila & Zanibbi, 2017).

The symbol layout tree was intended to encode PMML. Positional relations between math symbols are encoded by the SLT. This allows one to trace a path through the tree and represent that path using the symbols associated to the edges and nodes. These "paths" have a special dictionary that represents each symbol or operation. These paths are tokenized and indexed as separate words in the inverted index. These encodings are what are used to calculate the tf-idf weights.

The advantage of SLTs is that they capture the left to right order within the formula. However, not all operations occur in a left to right manner. This is where operator trees can capture more information. OPTs, when traversed starting at the lowest level of leaves, will output the correct sequence of calculations with a depth-first search. In this way, they are more representative of CMML representations as opposed to PMML.

2.1c Tangent-S and Tangent-CFT

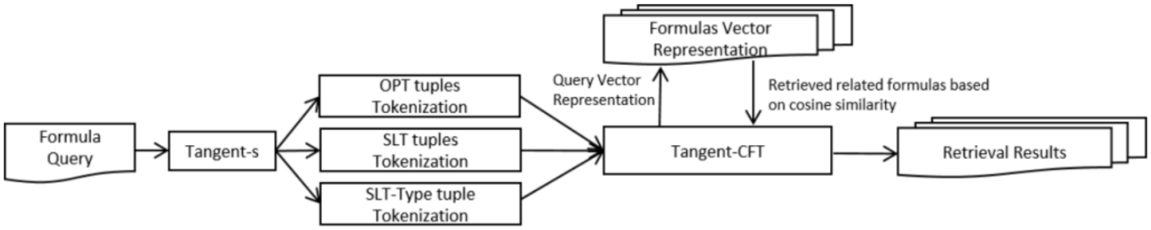

Formulas Vector Representation
Query Vector Representation
Retrieved related formulas based on cosine similarity
Formula Query
Tangent-s
OPT tuples Tokenization
SLT tuples Tokenization
SLT-Type tuple Tokenization
Tangent-CFT
Retrieval Results

Figure 3: Tangent-CFT[3]

Figure 3 shows the general structure of the current baseline standard for building a Math Formula Embedding model. Tangent-s is the system that builds the SLT and OPT tree representation from the MathML representation. A single formula is first converted into its tree representations. Then, for each pair of symbols (each pair of connected nodes in the trees) a tuple is created using symbols to represent the math operations. In this way, each formula is tokenized and sharded into a set of tuples representing all the operations. These sets of tuples allow you to perform a pseudo-shingle-like approach to break down the function into segments and use these segments as indexes in the dictionary. Upon creating these pairwise tuples from the trees, a simple expression such as $x - y^2 = 0$ is now broken down into many segments because each

[3] https://www.cs.rit.edu/~rlaz/files/Mansouri_ICTIR2019.pdf

pair of expressions or each pair of connected nodes in the trees are encoded. Mansouri produces over 15 tuples just for this simple function $x - y^2 = 0$. Each of these tuples can be further tokenized where each symbol/keyword in the tuple represents its own word in the index. In this way, we are basically character n-grammifying the math formula using its tree representations, where a character is a symbol from the tree-generated tuple representation.

Tangent-CFT builds upon the index system generated by Tangent-S. Tangent-CFT is the system that reads these tree-to-tuple representations and combines them to form a single formula vector. Individual vectors are computed for each of the tree representations via standard systems such as the tf-idf approach. Tangent-CFT then performs a simple sum operation to add all these vectors together. Using this singular vector, cosine similarity is used to calculate document similarity.

3. Transformer Models

Now that we understand the fundamentals of the base systems of Math IR, I want to explain the recent success of Transformer models. In a transformer model, there are two main types of layers: the Encoder and Decoder. The encoder takes a sentence, tokenizes the sentence and outputs an embedding for the sentence. The embedding is essentially a large vector representation of each word or token read by the Encoder layer. A decoder model is a sequence-to-sequence generator. The decoder takes in a sequence and outputs a sequence that the model believes should follow the previous sequence. Both models require two phases of training. Pre-training is when you train the model on a large corpus of data, where it learns from the self-labeled data you input. Fine-Tuning is another phase of training, where the weights of pre-training are reused. Fine-Tuning typically does not necessitate a large corpus. Its purpose is to help your model learn your specific domain, given it already has some pre-trained knowledge.

In recent years, transformer models have been increasingly used in the context of Information Retrieval. The embeddings produced by the encoder has been shown to be incredibly efficient for creating formula vector representations of a sentence. Because Transformer models such as BERT are architected to allow them to understand bi-directional context within text, they have proven quite effective for IR. Since BERT’s introduction, many studies have shown the increase in results on standard IR tasks using BERT compared to other standard models.

3a. CompuBERT

CompuBERT was one of the first attempts in applying a Transformer based model on Math IR. The basic architecture of CompuBERT is outlined in Figure 4.

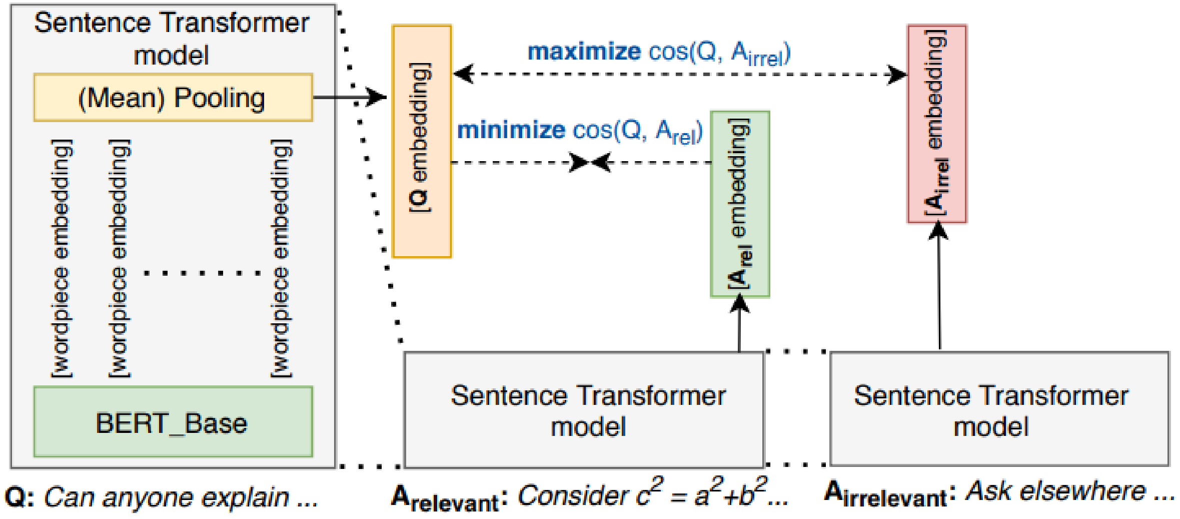


Figure 4: CompuBERT (Novotny et al, 2019)

Novotny describes CompuBERT, which was designed as one of the first Math IR transformer models. CompuBERT takes a BERT base model and uses its encoder to create Query and Answer embeddings. Both query and answer pairs are fed into the model, along with a label indicating relevance or irrelevance between the two. An embedding is output for both the query and document. If the document query pair is considered relevant, the model attempts to minimize the cosine distance between their embeddings by adjusting its weights. Otherwise, if the document is irrelevant, the model attempts to maximize the cosine distance between the two embeddings. Overall, this model did not have too much success by trec eval metrics but laid out the foundation for more successful models in the future.

3b. Albert and BERT for ARQMath 3

In ARQMath 2 and 3, more transformer-based IR systems were introduced. Reusch introduced a new training method for Transformer models that was geared toward Math documents. Reusch points out that models in the past were not pre-trained on any math corpus and thus, were not achieving results up to par with other core models such as the augmented Tangent-CFT models. Instead, they proposed a new method of pre-training.

Starting with a base ALBERT, BERT and RoBERTa model (different variations of the BERT base model), they fully pre-train the models on some math corpus. They start with training only on math formulae. First, they created special tokens to add to the model that represent different operations in math formulae. They add these tokens to improve the tokenization of sentences before its fed into the model. One thing I noticed, while training

BERT models, was that the standard tokenizers would break up math operations. For instance, the term ‘\\frac’ would get tokenized into ‘\\’ and ‘frac’. To fix this issue, Reusch created 512 special tokens. After adding these tokens, ‘\\frac’ is tokenized to ‘\\frac’.

Next, the math formulae are sharded into parts and the BERT models is fed these sharded parts of formulae as question-and-answer pairs. In this way, the BERT model encoder improves its ability to create embeddings for math formulae. They also attempted other clever methods such as mixing up the input training data for the model by matching more q+a pairs that were classified as irrelevant. This allowed the model to better learn not just what q+a pairs were relevant but irrelevant. Only after this pre-training process, did they fine-tune their model on a subset of the Math Stack Exchange Dataset. Model cards and access to reuse their models are provided on HuggingFace[4].

a. MathBERT

MathBERT was another model that I tried to implement for the ARQMath Task 1 results. MathBERT was the first pre-trained model for mathematical formula understanding (Peng et al, 2021). MathBERT like the Albert/RoBERTa model above, was trained specifically on a math dataset. It was pretrained using OPTs and formulas. MathBERT was trained on text representation, latex representation and OPT tuples of math formulae. They also applied three pretraining techniques: Masked Language Modeling where random tokens are masked to improve model robustness, Context Correspondence Prediction to account for context in math equations, and Masked Substructure Prediction to incorporate the structural features of OPTs (Peng et al, 2021). Their pretrained model is provided in their hugging face model card.[5]

b. MPNet

Masked and Permuted Pre-training for Language Understanding (MPNet) was the last pretrained model I used in my calculation. Due to the fact that this model had the highest pre-trained model score on various standard NLP datasets, I wanted to test out its performance on Math IR. MPNet’s main novelty is that it was trained in a way where it can better understand not only bidirectional context of tokens (BERT’s novelty) but also inherent dependency among tokens introduced by XLM (Song et al, 2020). I assumed these features could potentially improve the model’s ability to capture more information from a math formula. Hence, I decided to pretrain this model on a large corpus of Math IR data (namely the entire Math Stack Exchange dataset) as well as modify the tokenizer in the same way done in Albert for ARQMath-3.

[4] https://huggingface.co/AnReu/albert-for-arqmath-3

[5] https://huggingface.co/tbs17/MathBERT

c. ColBERT

ColBERT introduced a novel method of computing query-document similarities using BERT based models. They introduced a concept called contextualized late interaction (Khattab & Zaharia, 2020). One of the downsides of using a BERT based Q+A model is that to find a ranked list of relevant documents, queries must be scored against every document. Every query and document pair must be fed into the BERT model to produce an accurate classification of relevance. This introduces heavy computation that is often limited by what GPU compute engine you have access to. You can imagine given a document set of a million documents, computing embeddings a million times per query is not cost effective. What ColBERT introduced was a *late interaction* paradigm, where document and query embeddings were pre-computed and stored on disk. Once you have these pre-computed embeddings, you can perform a simple, cost-effective measure of similarity across queries and documents. You eliminate the need to run both query and document into the BERT model at the time of query. You perform one cost-intensive offline computation for the query and document embeddings. Once these are stored, you can use those precomputed embeddings for retrieval at a fraction of the cost. They showed that their method decreased query latency by three to four orders of magnitude.

The model I use incorporates this idea of contextualized late interaction. In all previous iterations of BERT in Math IR, query and document embeddings are computed via the standard cost-ineffective method. Feeding every query, every document in the corpus to compute a rank. The Albert for ARQMath 3 model for instance took 3 hours to compute query results. The 3 hours were needed, despite using a highly modern cluster of GPUs (8 A100 GPUs), which are the industry gold standard for GPU processing. Using my local computer, which uses a single RTX 2080 ti (about four to five times slower than a single A100), I compute results for each query at an average speed of 10 seconds using my base method and 1 minute using my Cross Encoder method (to be discussed in the next section).

d. Cross-Encoder Re-ranker

The last technique attempted is called Cross-Encoder reranking. Cross-encoders are similar to bi-encoders but rather than taking each query and document separately, the cross-encoder takes both query and document simultaneously and produces a single classification score for the query

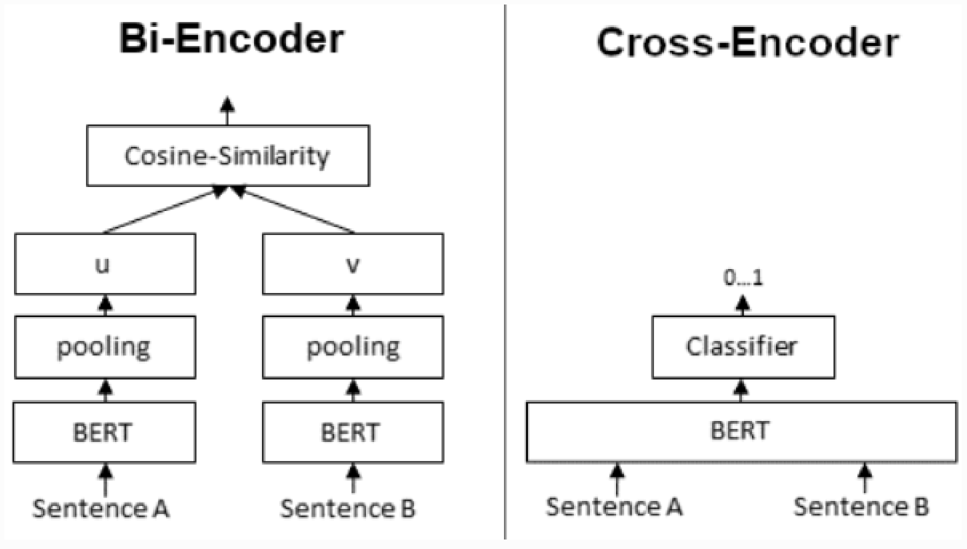


Figure 5: Bi-Encoders take cosine similarity between separately encoded sentences. Cross-encoders combine the two sentences and feed them simultaneously into the model.[5]

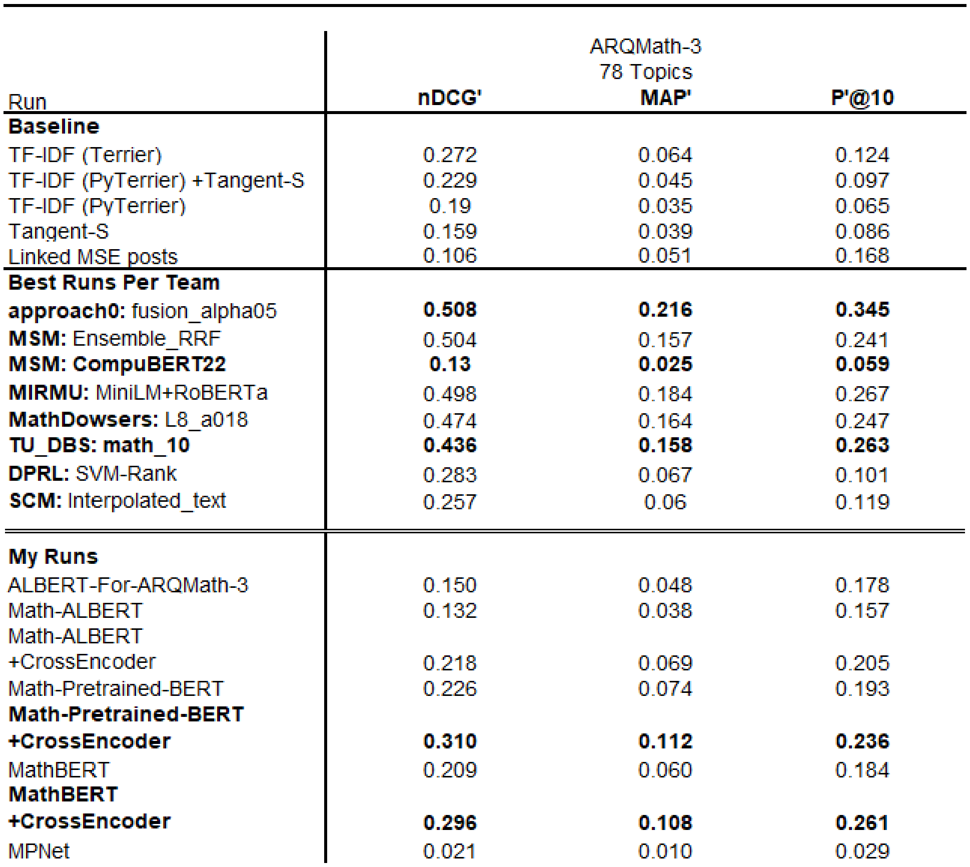
and document pair. Figure 5 explains the architecture. The basic justification for this method is as follows. Given the initial BERT models were initially trained as cross-encoders, there was a need to account for the fact that using the late interaction approach inherently prevents feeding both query and document as a single sample to the model at retrieval time. To address this concern, a cross-encoder re-ranker was implemented.

The basic structure is as follows. First, the model computes document and query embeddings offline as described in late interaction. Due to memory constraints, the model shards the query embeddings into chunks of sentence embeddings. These chunks are all stored on disk. At retrieval time, for each query, a query embedding is computed and the model computes cosine similarity with each chunk of document embeddings. The model returns the top-k*n documents for each of these chunks. This means my *late interaction* step will return $kn * C$ documents, where C is the number of chunks. In the final experiment, the model takes 3000 documents from each chunk, given the submission requires 1000 top documents per query. I then apply my cross-encoder to re-rank this reduced subset of documents and finally return the top-k of documents from the new scores. The benefit of this method is that rather than applying a cross encoder on the entire set of 1.5 million documents, I only apply the cross encoder on 9000 documents. Applying this cross-encoder method significantly improved performance but increased runtime cost from 10 seconds per query to a minute per query.

[6] https://www.sbert.net/examples/applications/cross-encoder/README.html

4. Results

Below are my results for Task 1 of the ARQMath conference. I provide scores relative to the scores from ARQMath-3.



		ARQMath-3 78 Topics	
Run	**nDCG'**	**MAP'**	**P'@10**
Baseline			
TF-IDF (Terrier)	0.272	0.064	0.124
TF-IDF (PyTerrier) +Tangent-S	0.229	0.045	0.097
TF-IDF (PyTerrier)	0.19	0.035	0.065
Tangent-S	0.159	0.039	0.086
Linked MSE posts	0.106	0.051	0.168
Best Runs Per Team			
approach0: fusion_alpha05	**0.508**	**0.216**	**0.345**
MSM: Ensemble_RRF	0.504	0.157	0.241
MSM: CompuBERT22	**0.13**	**0.025**	**0.059**
MIRMU: MiniLM+RoBERTa	0.498	0.184	0.267
MathDowsers: L8_a018	0.474	0.164	0.247
TU_DBS: math_10	**0.436**	**0.158**	**0.263**
DPRL: SVM-Rank	0.283	0.067	0.101
SCM: Interpolated_text	0.257	0.06	0.119
My Runs			
ALBERT-For-ARQMath-3	0.150	0.048	0.178
Math-ALBERT	0.132	0.038	0.157
Math-ALBERT +CrossEncoder	0.218	0.069	0.205
Math-Pretrained-BERT	0.226	0.074	0.193
Math-Pretrained-BERT +CrossEncoder	**0.310**	**0.112**	**0.236**
MathBERT	0.209	0.060	0.184
MathBERT +CrossEncoder	**0.296**	**0.108**	**0.261**
MPNet	0.021	0.010	0.029

Table 1: Results for ARQMath-3

Table 1 shows the results of my experiments. The first model (ALBERT-for-ARQMath-3) is the same model that the TU_DBS team used in their submission that achieved a score of 0.436 nDCG'. Also, the MPNet model that I attempted to pre-train on the Math Stack Exchange data performed the worst. This most probably had to do with how the data was preprocessed for my training. The training did not include SLTs and OPTs, so the model was not able to learn math encodings in the same way as the other highest performing models.

For the other BERT-based models, after applying the late interaction step, the query results perform notably worse than their counterparts (TU_DBS' full list of model scores can be found in the appendix Table 2). Interestingly, after applying late interaction to the same math_10 TU_DBS model, performance worsened significantly. However, all the BERT based models still significantly outperform the CompuBERT model – from which I based my initial architecture – and most of the baseline models especially in terms of P'@10.

4.1 Model Performance Analysis

The best models were the Math-Pretrained-BERT+CE and the MathBERT+CE models. Both models notably outperform the standard baseline models and have performance comparable to many of the other submissions. In particular, the precision @ 10 had a very high score, comparable with the other runs by the top teams.

The results show that the models tended to underperform in terms of nDCG' and MAP' but retain high P'@10 scores. My conclusion is that applying late interaction does not significantly degrade performance of queries for the top ranked answers. However, lower ranked answers tend to lose efficacy. I anticipate the model can retrieve the most relevant documents but struggles to retrieve documents that are relevant but with a lower relevancy score. This is one area I would like to apply further testing to my hypothesis. The Math-Pretrained-BERT+CrossEncoder model showed a $P'@5 = 0.249$, $P'@10 = 0.236$ and a $P'@50 = 0.184$. Initial analysis leads me to believe the model's top 5-10 answers seem to be the most accurate with model performance decreasing as 'k' increases. This would also explain the lower Mean Average Precision score despite a high precision @ 10.

Although, model performance degraded because of applying late interaction, model latency significantly improved. According to Reusch et al. (the TU_DBS team), their model's evaluation took 3 hours to complete, using 8 A100 GPUs. My model's evaluation took 20 minutes to complete using a single RTX 2080 ti GPU. The model + CrossEncoder took 2 hours to complete using the same single GPU. To help contextualize this vast increase in performance, trying to run the evaluation used by the TU_DBS team on my single GPU would have taken 10-15 hours to complete.

One observation is that the Math-Pretrained-BERT and MathBERT models without cross-encoders may have had worse scores overall, but they take merely 10 seconds to compute results per query. Although their nDCG' and MAP' suffer, the P'@10 scores remain quite similar. Due to significant differences in latency of retrieval, one may prefer the non-CE implemented models despite its drop in overall performance.

5. Conclusion

Applying late interaction to Transformer based models in Math IR may impact performance of the overall IR system; however, it is worth noting if brought into production, will the tradeoff of query latency be worth the increase in performance. For queries where users expect to only receive the top 10 or less documents, this late interaction BERT-based model would be preferred over the more accurate but higher latency transformer models. Overall, the decrease in performance partially has to do with how the pretrained Math models were trained. The models themselves were trained as classifiers for relevant and irrelevant documents. They were not intended to be used as encoders. Also, the preprocessed encodings used for the pretraining were not the same preprocessed encodings that my experiment used in fine-tuning. In the future, more research must be done regarding how to efficiently apply late interaction for math IR queries. Further study must be done to pre-train the models to efficiently function as sentence encoders and not just classifiers. Additionally, it will be useful to evaluate how the model will improve if OPTs and SLTs are included in the tokenization during fine-tuning and pretraining.

References

Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In SIGIR

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2021. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. arXiv:2112.01488

Pankaj Dadure, Partha Pakray, and Sivaji Bandyopadhyay. 2021. BERT-based embedding model for formula retrieval. In CLEF.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805.

Kenny Davila and Richard Zanibbi. 2017. Layout and semantics: Combining representations for mathematical formula search. In SIGIR.

Shuai Peng, Ke Yuan, Liangcai Gao, and Zhi Tang. 2021. MathBERT: A pre-trained model for mathematical formula understanding. arXiv:2105.00377.

Anja Reusch, Maik Thiele, and Wolfgang Lehner. 2021a. An ALBERT-based similarity measure for mathematical answer retrieval. In SIGIR.

B. Mansouri, V. Novotný, A. Agarwal, D. W. Oard, R. Zanibbi, Overview of ARQMath-3 (2022): Third CLEF lab on Answer Retrieval for Questions on Math, in: CLEF 2022, volume 13390 of LNCS, Springer, 2022.

Anja Reusch, Maik Thiele and Wolfgang Lehner. 2022. Transformer-Encoder and Decoder Models for Questions on Math, in: CLEF Vol-3180;

Kaitao Song, Xu Tan 0003, Tao Qin, Jianfeng Lu, Tie-Yan Liu. MPNet: Masked and Permuted Pre-training for Language Understanding. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, Hsuan-Tien Lin, editors, Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual. 2020.

Mansouri, B., Rohatgi, S., Oard, D.W., Wu, J., Giles, C.L., Zanibbi, R.: Tangent-CFT: An Embedding Model for Mathematical Formulas. In: Proceedings of the 2019 ACM SIGIR international conference on theory of Information Retrieval. pp. 11–18 (2019)

Vit Novotny, Petr Sojka, Michal Stefanik, and David Luptak: 2020: Three is Better than One Ensembling Math Information Retrieval Systems; CLEF 2020.

Appendix

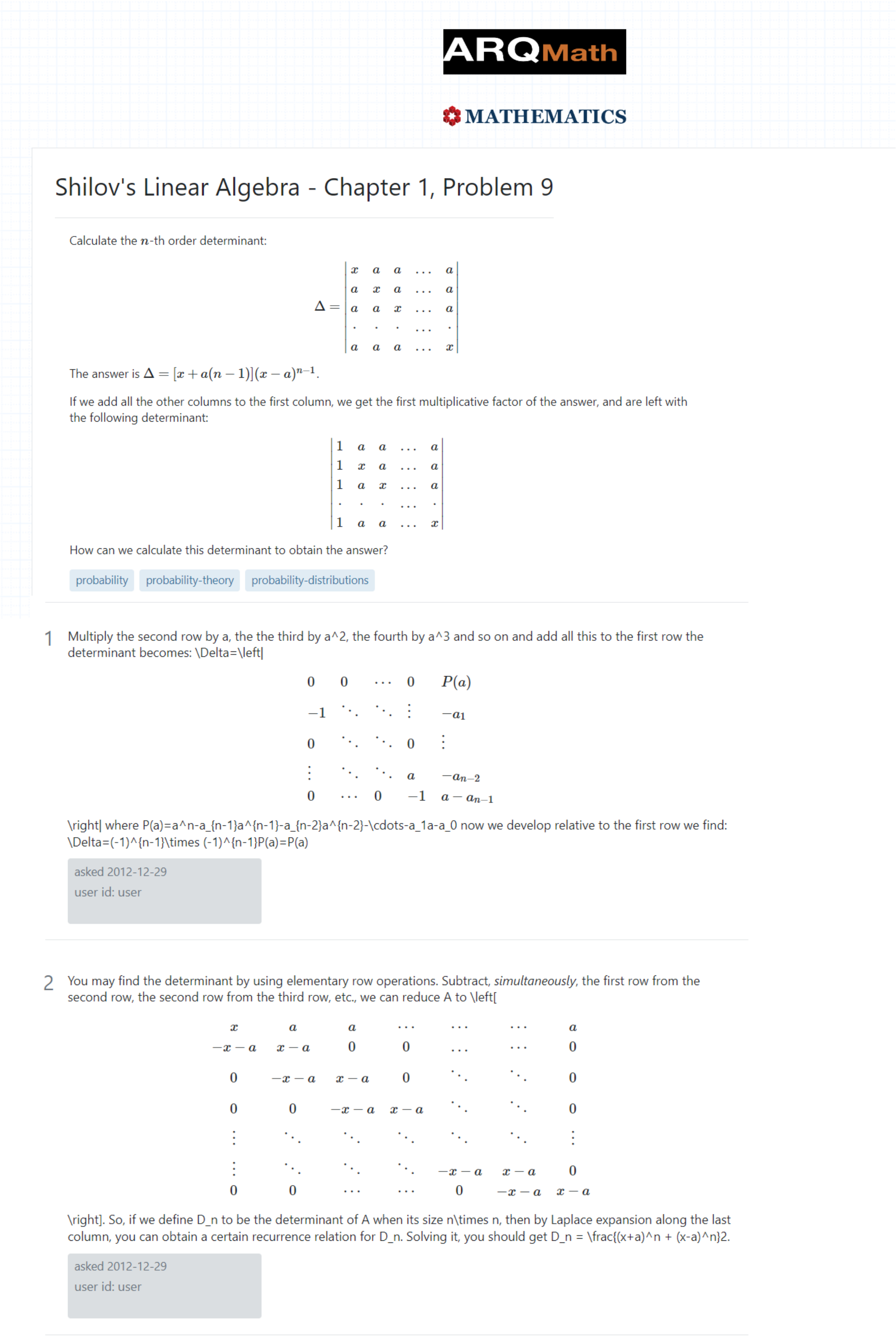
Table 2

List of table results for all Task 1 submissions for ARQMath-3.

		ARQMath-3 78 Topics	
Run	**nDCG'**	**MAP'**	**P'@10**
Baseline			
TF-IDF (Terrier)	0.272	0.064	0.124
TF-IDF (PyTerrier) +Tangent-S	0.229	0.045	0.097
TF-IDF (PyTerrier)	0.19	0.035	0.065
Tangent-S	0.159	0.039	0.086
Linked MSE posts	0.106	0.051	0.168
approach0			
fusion_alpha05	**0.508**	**0.216**	**0.345**
fusion alpha03	0.495	0.203	0.317
fusion alpha02	0.483	0.195	0.305
rerank nostemer	0.418	0.172	0.309
a0porter	0.397	0.159	0.271
MSM			
Ensemble RRF	0.504	0.157	0.241
BM25 system	0.396	0.122	0.194
BM25_TfIdf system	0.396	0.122	0.194
TF-IDF	0.28	0.064	0.081
CompuBERT22	0.13	0.025	0.059
MIRMU			
MiniLM+RoBERTa	0.498	0.184	0.267
MiniLM +MathRoBERTa	0.496	0.181	0.273
MiniLM_tuned +MathRoBERTa	0.494	0.178	0.262
MiniLM_tuned +RoBERTa	0.472	0.165	0.244
MiniLM+RoBERTa	0.35	0.107	0.159
MathDowsers			
L8 a018	0.474	0.164	0.247
L8 a014	0.468	0.155	0.237
L1on8 a030	0.467	0.159	0.236
TU_DBS			
math 10	0.436	0.158	0.263
Khan SE 10	0.426	0.154	0.236
base 10	0.423	0.154	0.228
roberta 10	0.413	0.15	0.226
math 10 add	0.379	0.149	0.278
DPRL			
SVM-Rank	0.283	0.067	0.101
RRF-AMR-SVM	0.274	0.054	0.022
QQ-QA-RawText	0.245	0.054	0.099
QQ-QA-AMR	0.185	0.04	0.091
QQ MathSE-AMR	0.178	0.039	0.081
SCM			
interpolated_text +position_word 2vec tangentl	0.257	0.06	0.119
joint word2vec	0.249	0.059	0.106
joint_tuned roberta	0.249	0.059	0.105
joint_positional word2vec	0.248	0.059	0.105
joint roberta base	0.188	0.04	0.077

Query Results

Here I show an HTML document templated by the ARQMath team. It shows the best performing query among all the official query list. I show the first 5 answers to the query.



ARQMath

MATHEMATICS

Shilov's Linear Algebra - Chapter 1, Problem 9

Calculate the n-th order determinant:

$$\Delta=\begin{vmatrix} x & a & a & \dots & a \\ a & x & a & \dots & a \\ a & a & x & \dots & a \\ \cdot & \cdot & \cdot & \dots & \cdot \\ a & a & a & \dots & x \end{vmatrix}$$

The answer is $\Delta=[x+a(n-1)](x-a)^{n-1}$.

If we add all the other columns to the first column, we get the first multiplicative factor of the answer, and are left with the following determinant:

$$\begin{vmatrix} 1 & a & a & \dots & a \\ 1 & x & a & \dots & a \\ 1 & a & x & \dots & a \\ \cdot & \cdot & \cdot & \dots & \cdot \\ 1 & a & a & \dots & x \end{vmatrix}$$

How can we calculate this determinant to obtain the answer?

probability probability-theory probability-distributions

1 Multiply the second row by a, the the third by a^2, the fourth by a^3 and so on and add all this to the first row the determinant becomes: \Delta=\left|

$$\begin{matrix} 0 & 0 & \cdots & 0 & P(a) \\ -1 & \ddots & \ddots & \vdots & -a_1 \\ 0 & \ddots & \ddots & 0 & \vdots \\ \vdots & \ddots & \ddots & a & -a_{n-2} \\ 0 & \cdots & 0 & -1 & a-a_{n-1} \end{matrix}$$

\right| where P(a)=a^n-a_{n-1}a^{n-1}-a_{n-2}a^{n-2}-\cdots-a_1a-a_0 now we develop relative to the first row we find: \Delta=(-1)^{n-1}\times (-1)^{n-1}P(a)=P(a)

asked 2012-12-29
user id: user

2 You may find the determinant by using elementary row operations. Subtract, *simultaneously*, the first row from the second row, the second row from the third row, etc., we can reduce A to \left[

$$\begin{matrix} x & a & a & \cdots & \cdots & \cdots & a \\ -x-a & x-a & 0 & 0 & \cdots & \cdots & 0 \\ 0 & -x-a & x-a & 0 & \ddots & \ddots & 0 \\ 0 & 0 & -x-a & x-a & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & -x-a & x-a & 0 \\ 0 & 0 & \cdots & \cdots & 0 & -x-a & x-a \end{matrix}$$

\right]. So, if we define D_n to be the determinant of A when its size n\times n, then by Laplace expansion along the last column, you can obtain a certain recurrence relation for D_n. Solving it, you should get D_n = \frac{(x+a)^n + (x-a)^n}2.

asked 2012-12-29
user id: user

3 A bit of reducing: \left(

$$\begin{matrix} a & 1 & 1 \\ 1 & a & 1 \\ 1 & 1 & a \end{matrix}$$

\right), subtract the third row from the other two: \left(

$$\begin{matrix} a-1 & 0 & 1-a \\ 0 & a-1 & 1-a \\ 1 & 1 & a \end{matrix}$$

\right), then add the first two columns to the third one: \left(

$$\begin{matrix} a-1 & 0 & 0 \\ 0 & a-1 & 0 \\ 1 & 1 & a+2 \end{matrix}$$

\right).

Now you can easily obtain the determinant.

asked 2012-12-29
user id: user

4 It's the determinant of a Vandermonde matrix. The standard argument to compute the determinant is to write f(x) =

$$\begin{vmatrix} 1 & 1 & \cdots & 1 \\ a_0 & a_1 & \cdots & x \\ a_0^2 & a_1^2 & \cdots & x^2 \\ \vdots & \vdots & \ddots & \vdots \\ a_0^n & a_1^n & \cdots & x^n \end{vmatrix}$$

. This is a polynomial in x of degree n, and by the properties of determinants (the determinant of a matrix with two columns equal is zero), you have that a_0, a_1, \dots, a_{n-1} are roots. Therefore f(x) = A \cdot (x-a_0) \cdot (x-a_1) \cdot\ldots \cdot (x- a_{n-1}). Now looking at the matrix, you see that the coefficient A of x^{n} is a similar determinant of size one less. Use induction to conclude.

asked 2012-12-29
user id: user

5 More generally, consider the n\times n matrix M(a)=

$$\begin{bmatrix} a & a & a & \dots & a \\ a & a & a & \dots & a \\ \vdots & \vdots & \ddots & \dots & \vdots \\ a & a & a & \dots & a \end{bmatrix}$$

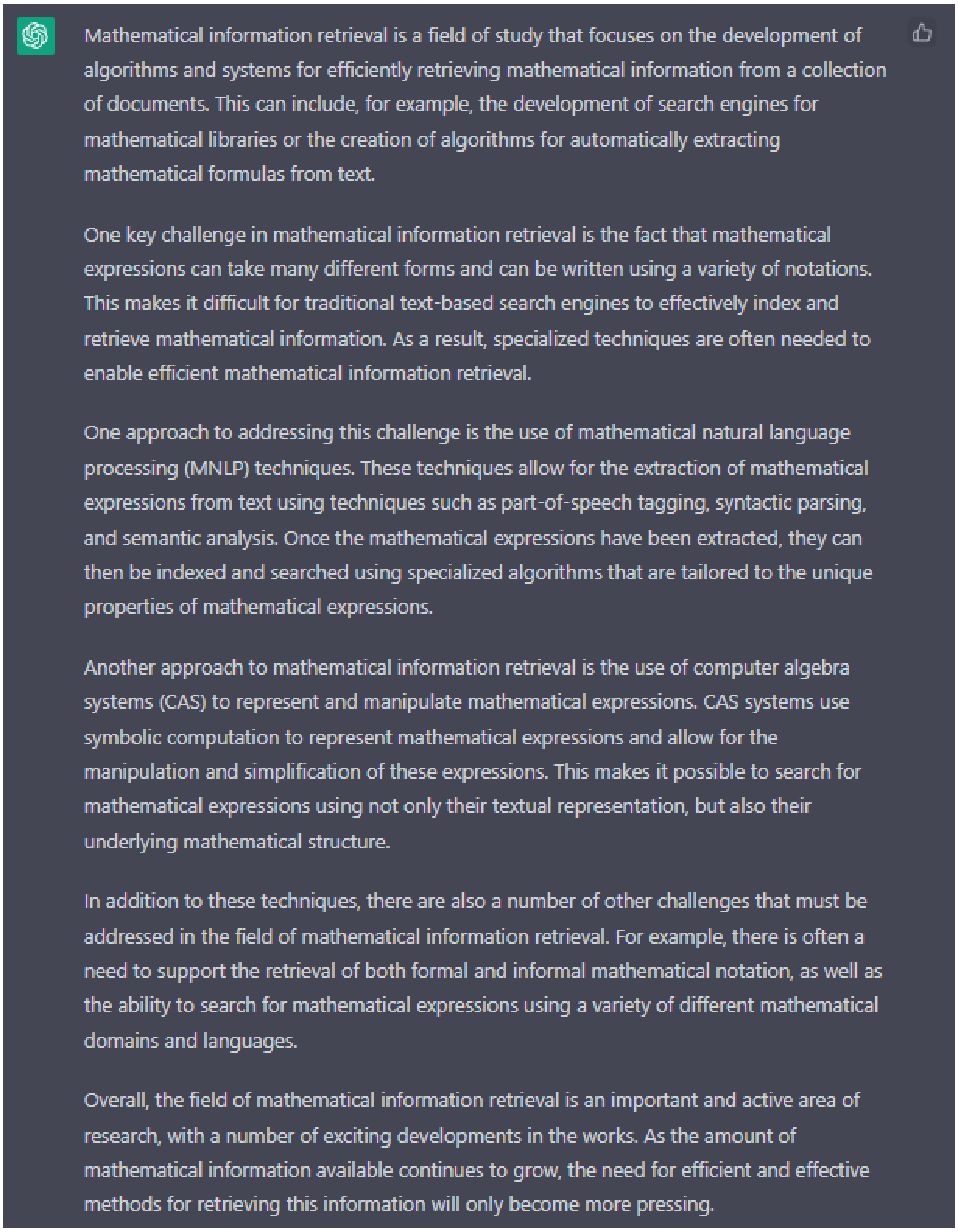
The characteristic polynomial of M(a) is \det(M(a)-XI)=(0-X)^{n-1}(na-X) Can you see why?

What value can you give to X in order to have your given matrix?

asked 2012-12-29
user id: user

Just For Fun

Below I added the text returned from the ChatGPT model when I asked it to write a paper on Math IR. All credit OpenAI and their implementation. No copyright intended.



Mathematical information retrieval is a field of study that focuses on the development of algorithms and systems for efficiently retrieving mathematical information from a collection of documents. This can include, for example, the development of search engines for mathematical libraries or the creation of algorithms for automatically extracting mathematical formulas from text.

One key challenge in mathematical information retrieval is the fact that mathematical expressions can take many different forms and can be written using a variety of notations. This makes it difficult for traditional text-based search engines to effectively index and retrieve mathematical information. As a result, specialized techniques are often needed to enable efficient mathematical information retrieval.

One approach to addressing this challenge is the use of mathematical natural language processing (MNLP) techniques. These techniques allow for the extraction of mathematical expressions from text using techniques such as part-of-speech tagging, syntactic parsing, and semantic analysis. Once the mathematical expressions have been extracted, they can then be indexed and searched using specialized algorithms that are tailored to the unique properties of mathematical expressions.

Another approach to mathematical information retrieval is the use of computer algebra systems (CAS) to represent and manipulate mathematical expressions. CAS systems use symbolic computation to represent mathematical expressions and allow for the manipulation and simplification of these expressions. This makes it possible to search for mathematical expressions using not only their textual representation, but also their underlying mathematical structure.

In addition to these techniques, there are also a number of other challenges that must be addressed in the field of mathematical information retrieval. For example, there is often a need to support the retrieval of both formal and informal mathematical notation, as well as the ability to search for mathematical expressions using a variety of different mathematical domains and languages.

Overall, the field of mathematical information retrieval is an important and active area of research, with a number of exciting developments in the works. As the amount of mathematical information available continues to grow, the need for efficient and effective methods for retrieving this information will only become more pressing.