Multi-Stage Pre-Training for Math-Understanding: μ^2(AL)BERT

Anonymous ACL submission

Abstract

Understanding mathematics requires not only comprehending natural language, but also mathematical notation. For mathematical language modeling, current pre-training methods for transformer-based language models which were originally developed for natural language need to be adapted. In this work, we propose a multi-stage pre-training scheme including natural language and mathematical notation that is applied on ALBERT and BERT, resulting in two models that can be fine-tuned for downstream tasks: μ^2ALBERT and μ^2BERT. We show that both models outperform the current state-of-the-art model on Answer Ranking. Furthermore, a structural probing classifier is applied in order to test whether operator trees can be reconstructed from the models' contextualized embeddings.

1 Introduction

Transformer-based Language Models (LM) have not only a high impact on all domains in Natural Language Understanding, but also on related fields that besides natural language try to model artificial languages such as programming code or mathematical notation written in LaTeX. Since pre-training of these models is by its nature optimized for natural language, for artificial languages the training needs to be adapted. This work focuses on scientific literature where coherent natural language text is discontinued by formulas written using LaTeX. Since models like BERT (Devlin et al., 2018) rely on coherent text, this phenomenon might not be handled correctly. Therefore, we propose a pre-training scheme consisting of multiple stages to adapt transformer-based language models to domains where mathematical notation is mixed with coherent text. In order to demonstrate its improvements over the vanilla models, we apply the training scheme to BERT and ALBERT.

Because both models already posses powerful natural language modeling capacities, it is not necessary to train them from scratch on our new domain and, hence, we start from these general models. Recent research has shown that pre-training on formula-sentence order prediction improves performance on formula-only and NL-only examples, but in a task where both natural language and mathematical notation occurs, the model still performs poorly (Reusch et al., 2021). Hence, we apply both methods to our model: Starting from the NL-pre-trained published checkpoint, we first train the model on a formula-NL order prediction task, then using the standard sentence order prediction task. To improve the models understanding of formulas itself, we also add a Mathematical Modeling stage, where the model is trained on MLM and SOP only on formulas in LaTeX.

While some models for mathematical modeling rely on the standard sub-word tokenizer of BERT (Reusch et al., 2021), others add LaTeX tokens to the existing vocabulary (Jo et al., 2021). In this work, we also investigate, whether it is necessary to add tokens to the vocabulary or whether the sub-word tokenization might already work well enough.

Our evaluation is carried out in two different ways: We first evaluate our models on the downstream task answer ranking for mathematical questions using the ARQMath Lab 2020 data set. This task is especially useful, since is contains natural language text and formulas which need to be understood by the model in order to retrieve the right answer. Secondly, we analyze how well the models are able to reconstruct a parse tree of the mathematical formulas on their own. Here, we employ the structural probes introduced by Hewitt et al. for operator trees (Hewitt and Manning, 2019).

In total, the main contributions of this work are:

- We present a Multi-stage pre-training scheme for Transformer-based LM for Math-understanding and apply it on ALBERT and BERT: μ^2ALBERT and μ^2BERT

- We evaluate the need for extending the tokenizer by adding LaTeX tokens
- Finally, we train a probing classifier for the mathematical knowledge LMs can learning using only LaTeX input.

Our evaluation confirms the multi-stage scheme for ALBERT and BERT by outperforming the previous best model on the answer ranking task ARQMath 2020, as well as our decision to extend the models vocabulary by LaTeX tokens. Secondly, we show that mathematical knowledge in form of OPTs is extractable from the BERT's contextualized embeddings. For ALBERT, the results are not clear: While μ^2ALBERT improves over ALBERT base, it scores lower than BERT base that was not trained on mathematical notation.

The following work is structured as follows: First, we review work related to domain-adaptive pre-training in different domains and in Mathematical Language Modeling in particular, as well as research on the learned knowledge of transformer-based language models. Then we introduce the pre-training and our fine-tuning task using BERT and ALBERT. Sec. 4 presents the probing classifier while Sec. 5 details what data is used in each stage and provides information on the training process. Our results on the fine-tuning task and the structural probe are reported in Sec. 6. Sec. 7 concludes this work.

2 Related Work

The influence of domain-adaptive pre-training on BERT has been investigated by (Gururangan et al., 2020) who identified that this is particularly valuable when the domain vocabulary has a low overlap with the pre-training data. As a consequence, various models for different domains have been developed, such as BioBERT (Lee et al., 2020), ClinicalBERT (Alsentzer et al., 2019; Huang et al., 2019) or SciBERT (Beltagy et al., 2019) for scientific domains, or CuBERT (Kanade et al., 2020) and CodeBERT (Feng et al., 2020) for several programming languages. A notable difference between these models is that BioBERT, ClinicalBERT and CodeBERT use the original vocabulary that their base model was pre-trained on while SciBERT and CuBERT trained their own vocabulary specific to their domain. However, each of these models could demonstrate its improvements compared to the original models without domain specific pre-training.

BERT-based models for mathematical domains have also been studied with one example being MathBERT (Peng et al., 2021). Here, mathematical formulas in form of operator trees are used an input for pre-training. During the ARQMath Lab in 2020 and 2021, five teams submitted systems based on BERT, RoBERTa and SentenceBERT (Rohatgi et al., 2020; Novotnỳ et al., 2020; Rohatgi et al., 2021; Novotnỳ et al., 2021; Dadure et al., 2021; Mansouri et al., 2021) where the models were used without domain adaption for downstream tasks. Only (Reusch et al., 2021) pre-trained their submissions on mathematical documents. (Jo et al., 2021) fine-tuned a BERT model for notation prediction tasks based on scientific documents. They enlarged the vocabulary of BERT by additional LaTeX tokens.

However, little is known so far what BERT-based models learn about mathematics. In contrast, their learning capacities on natural language received large attention in recent research (for a survey see (Rogers et al., 2020)). Several probes and classifiers were employed to analyze whether BERT captures grammatical structures like dependency or constituency trees (Tenney et al., 2019; Hewitt and Manning, 2019; Coenen et al., 2019) or which layer attends to which linguistic feature (Clark et al., 2019). Visual frameworks like bertviz by (Vig, 2019) support the analysis of BERT's inner working by visualizing the attention weights of trained models. Also ALBERT was shown to capture part-of-speech tags in different places as reported by (Chiang et al., 2020), but most studies were performed using BERT.

3 Model

In this section, we will first review how the BERT and ALBERT models we employ are pre-trained and fine-tuned.

BERT and ALBERT are both transformer language models, which share a similar architecture. Both base models consist of 12 transformer encoder layers. BERT has 12 separate layers, while ALBERT's layers share their parameters resulting in a lower number of parameters being updated during training. The second difference between BERT and ALBERT are the embeddings: BERT uses embeddings of the dimension 756. ALBERT's initial embeddings have a dimensionality of 128.

The models as originally published are trained using two pre-training tasks which differ slightly.

Both are trained using the Masked Language Model task and a sequence classification task.

Pre-training is performed on a sentence-level granularity. Each sentence S is split into tokens: $S = w_1 w_2 \cdots w_N$. Before inputting the sentence into the model, each token w_i is embedded using a sum of three different embeddings, the word embedding t_i, the position embedding p_i, and the segment embedding s_i in order to discern between the first and the second segment when the model is presented e.g., two sentences as for the SOP task. The segment embeddings will also help our model to differentiate between the question and answer as the two segments later. All three embeddings are added up to form the input embedding E_i for each token: $E_i = t_i + p_i + s_i$. In order to obtain a representation of the entire input, the sentence S is prepended with a classification token $w_S = \langle CLS \rangle$. Between both segments and at the end of the sentence, the $\langle SEP \rangle$ token is placed. If the sentence is shorter than 512 tokens, the sequence is padded, otherwise, it will be truncated to 512 tokens.

3.1 Masked Language Model

The first pre-training task is the masked language model meaning tokens from the input sentence are randomly replaced by a $\langle MASK \rangle$ token, a different token, or is not modified at all. After embedding the input, it is fed into the language model (LM), consisting of 12 layers of transformer encoder blocks (Vaswani et al., 2017), resulting in a contextualized output embedding vector U_i for each input token:

$$CU_1 U_2 \cdots U_N = LM(E_{CLS} E_1 E_2 \cdots E_N),$$

where E_{CLS} and C denote the input and output embeddings of the $\langle CLS \rangle$ token. Afterwards, a classification layer is applied for the prediction of the original token from the input:

$$P(w_j|S) = softmax(U_i \cdot W_{MLM} + b_{MLM})_j,$$

where w_j is the j-th token from the vocabulary. This determines the probability that the i-th input token was w_j given the input sentence S. The weight matrix W_{MLM} and its bias b_{MLM} are only used for this pre-training task and are not used afterwards.

3.2 Segment Order Prediction

The next sentence prediction objective predicts whether the sentence given to the model as the first segment S_A appears in a text before the sentences given to the model as the second segment S_B (label 1) or not (label 0). This task is performed as a binary classification using the output embedding C as its input:

$$p(label = i|S) = softmax(C \cdot W_{SOP} + b_{SOP})_i,$$

where the matrix W_{SOP} and the bias b_{SOP} are only used for the SOP and are not re-used otherwise.

The sentence order prediction pre-training task as introduced by (Lan et al., 2019) requires input split into sentences and asks the model to predict their order. Here, we do not only use sentences for segments S_A and S_B. Therefore, we rename the task to segment order prediction (SOP). Originally, BERT was trained on the next sentence prediction task (NSP), but it has been demonstrated that ALBERT's sentence order prediction results in a better performance on intrinsic and downstream tasks. When pre-training ALBERT and BERT, we apply SOP to both models as their second task even though BERT was originally pre-trained using NSP.

3.3 Fine-Tuning

We fine-tune ALBERT and BERT models on the down-stream task of answer ranking (AR). More specifically, a classifier is trained on top of the models' output to predict whether an answer $A = A_1 A_2 \cdots A_M$ is relevant to a question $Q = Q_1 Q_2 \cdots Q_N$.

We present the input string $\langle CLS \rangle Q_1 Q_2 \cdots Q_N \langle SEP \rangle A_1 A_2 \cdots A_M$, with $\langle CLS \rangle$ denoting the classification token and $\langle SEP \rangle$ the separation token, to the model:

$$CU_1 U_2 \cdots U_N = LM(E_{CLS} E_1 E_2 \cdots E_{N+M+1}),$$

where E_i and E_{CLS} are the input embeddings for each input token and the $\langle CLS \rangle$ token, respectively, calculated as explained in Sec. 3.1. After the forward pass through the model, the output vector of the $\langle CLS \rangle$ token C is given into a classification layer:

$$p(label = i|Q, A) = softmax(C \cdot W_{AR} + b_{AR})_i,$$

where the label 1 denotes a matching or correct answer for the query and label 0 otherwise. During evaluation, the resulting probability of the classification layer for label 1, is the assigned similarity score s for the answer A to a question Q and is then used to rank the answers in the corpus:

$$s(Q, A) = p(label = 1|Q, A).$$

4 Mathematical Notation

In this work, we focus on formulas written in LaTeX which has several reasons: First, LaTeX is a popular typesetting tool used across all sciences. Hence, there exist a huge amount of training material. Prior research has experimented with using parse trees of formulas as an additional input during pre-training (Peng et al., 2021). However, this poses several issues: The input needs to be valid LaTeX code, for which the necessary packages need to be available to the parser. The parser has to be fast since each formula in the entire corpus must be parsed. And thirdly, we would restrict the learned knowledge to the knowledge that is left after parsing. By using a parser, information contained in the original LaTeX code could be missing in the result. Furthermore, it was already shown that BERT is able to learn grammatical structure of natural languages which could be extracted in the form of constituency and dependency trees (Tenney et al., 2019; Hewitt and Manning, 2019). If the model is able to learn these structures for which building parsers or grammars is much harder compared to mathematical notation, then it should be possible that it can also learn how structures in LaTeX work. Therefore, we advocate using LaTeX code directly. We analyze whether this is enough as input by applying a probing classifier on the learned contextualized embeddings of our models.

4.1 Structural Probe

The goal of the structural probe as introduced by (Hewitt and Manning, 2019) is to learn a matrix B, such that the distance $d_B(w_i, w_j) := \sqrt{(U_i - U_j)^\top B^\top B(U_i - U_j)}$ approximates a given tree distance d_T, i.e., the length of the path between the node of w_i and the node of w_j. B is learned by minimizing the loss function over all sentences S in the training corpus:

$$\min_B \sum_S \frac{1}{|S|^2} \sum_{i,j} |d_T(w_i, w_j) - d_B(w_i, w_j)|$$

Originally, Hewitt et al. applied the structural probe to demonstrate that dependency structures of the English language are to some extent contained in BERT's contextualized embeddings. In this work, we will train a structure probe to evaluate whether the models' inner workings have learned about mathematical structures, i.e., operator trees.

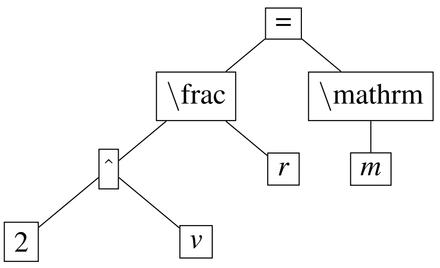


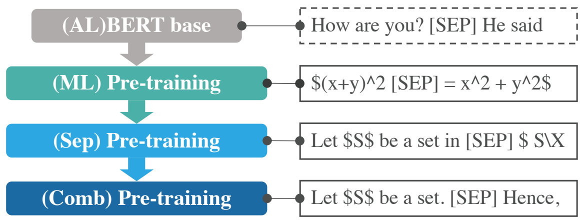
Figure 1: Operator Tree of the formula $\mathrm{m} = \frac{r}{v^2}$

4.2 Operator Trees

Formulas possess a hierarchical structure, which is encoded in Content Math ML, defining an operator tree (OPT). An example OPT for the equation $\mathrm{m} = \frac{r}{v^2}$ is shown in Fig. 1. Nodes of this tree representation can be individual or multiple symbols such as numbers, variables, text fragments indicating certain functions, fractions, radicals, LaTeX style expressions, or parentheses and brackets. This definition is similar to the one found in (Mansouri et al., 2019), but we added parenthesis and brackets to investigate the way our models capture open and closed bracket relationships. OPT edges indicate an operator-argument relationship between parent and child node. Left and right brackets and parentheses have each an edge to the parent of the tree inside them. LaTeX style expressions like `\mathbb` can be simply seen as an operator applied on the argument inside. Hence, the original OPT stays intact.

5 Experimental Setup

To improve how transformer-based language models understand mathematical notation, we want to investigate three questions in this paper: (1) Do we need to add LaTeX Tokens to the tokenizers? (2) Do all stages improve the models' performance? (3) Can we extract OPTs from the models' contextualized embeddings? In order to answer these questions, we train several models starting from the official checkpoints of BERT and ALBERT, respectively. To answer the first question, we train several ALBERT models using different stages, either with LaTeX tokens added to the models' and tokenizers' vocabulary or without. Question 2 is answered by evaluating the models using the mathematical answer ranking task of the ARQMath Lab 2020. We show that removing each of the stages presented in Sec. 5.1 degrades the downstream performance. Furthermore, the final pre-training scheme is applied to BERT which is also fine-tuned on AR. To



(AL)BERT base
How are you? [SEP] He said
(ML) Pre-training
$(x+y)^2 [SEP] = x^2 + y^2$
(Sep) Pre-training
Let S be a set in [SEP] $ S\X
(Comb) Pre-training
Let S be a set. [SEP] Hence,

Figure 2: The Stages for μ^2ALBERT and μ^2BERT

answer the third question, a probe is trained on top of both the ALBERT and BERT models and evaluated on the task of reconstructing the OPTs from a given LaTeX string. Here, we apply the structural probe introduced Section 4.1.

For answering Question (1) and (2), in total 11 models had to be trained. To save computation time, we pre-trained these configurations each on a single run using the ALBERT architecture, where the model needs fewer weights and converges faster during training. Only the final pre-training scheme was than applied to BERT, which takes more computing resources for pre-training and fine-tuning.

In the following, we will introduce the data we used for pre-training, fine-tuning, and evaluation as well as details on our experimental setup.

5.1 Data

We train and evaluate our models on data provided by the ARQMath Lab 2020 which contains question and answer posts from the Q&A community Mathematics StackExchange[1]. Pre-training used posts from 2010 to 2018 with HTML syntax removed. We wrapped formulas in \$ signs as it usual in LaTeX. Since ALBERT and BERT need separated input for the Segment Order Prediction task, all data needs to be split in segments. Here, we employed three different methods (Comb), (Sep) and (ML): (Comb) splits the natural language (NL) data into sentences,leaving LaTeX as it is. Lines split by newlines in the LaTeX code also served as two separate segments. (Sep) splits NL in sentences as (Comb), but also before the beginning of a LaTeX formula. Here, the model should also learn whether the LaTeX code and NL can follow each other (NL-formula coherence). Each LaTeX string was also split by newlines. (ML) removes all NL text, leaving only LaTeX code. The code is split into segments at relation symbols such as '=' or '$\in$'[2]. This way, the models should learn whether two LaTeX segments belong together (inter-formula coherence). These different pre-training corpora form essentially our three stages (ML), (Comb)and (Sep) applied on top of the natural language pre-trained checkpoint shown in Fig. 2. We also experimented with other orderings of the stages, but there were no improvements in performance visible. The data sizes for each stage are as follows: (Comb) contains 18M sentences, (Sep) 37M segments and (ML) 3.6M formula segments. The data was tokenized by either the standard tokenizer of ALBERT or BERT, respectively or their tokenizer extended by 501 LaTeX tokens as noted in the result section. μ^2ALBERT and μ^2BERT apply the extended tokenizer by default.

The fine-tuning data for the AR task is identical to the data used by (Reusch et al., 2021). We pair each question from the ARQMath 2020 corpus with one correct answer (label 1) and with one incorrect answer (label 0). The correct answer is chosen randomly from the answers given to the original question. The incorrect answer is selected by chance from a question with a similar topic, i.e., at least one of the questions tags have to match the original question. This procedure yielded 1.9M examples, of which 90% were used for training. The evaluation of our models is performed in the same way as for the ARQMath Lab which provides 77 questions from 2019 with relevance judgment for several answers annotated using pooling. Each of these questions is paired with every answer from the corpus which are then provided as input to the model. The classification score for label 1 is used to rank the answers. We do not pre-filter the answers by the tags of their questions, but use all answers in the corpus. For each question from the evaluation corpus, the organizers of the ARQMath Lab 2020 also annotated whether answering a question depended on understanding mainly the questions formulas, the natural language text or both. During evaluation we will break down the performance of our models by these three categories.

Our structural probe is trained on formulas which were parsed to OPTs by a custom LaTeX parser written in Python adapted from the parser rules of the mathematical formula search engine Approach0 (Zhong and Zanibbi, 2019; Zhong et al., 2020). We selected 50k training examples by

[1]https://math.stackexchange.com

[2]The entire list can be found in our GitHub repository: <Link anonymized>

chance from the corpus of all formulas from ARQMath 2020; in the same way we used 10k formulas each for development and test set.

5.2 Metrics

The AR task is scored using three metrics: nDCG', mAP' and p@10' by applying the scripts provided by the ARQMath Lab. The structural probe is evaluated using UUAS (undirected unlabeled attachment score) which denotes the percentage of correctly identified edges in the predicted tree and the DSpr. which is determined by first calculating the Spearman correlation between the predicted distances d_B and the gold-standard distances d_T. These correlations are then averaged between all formulas of a fixed length. Finally, the average across formulas of lengths 5–50 is reported as DSpr. We decided to include both metrics since it was shown that their scores can result in opposite trends (Hall Maudslay et al., 2020).

5.3 Setup

We trained all models using eight A100 GPUs with 40 GB GPU memory. For pre-training a batch size of 16 samples per GPU was used. Training ALBERT on stage (Comb) and (Sep) took 24h, which corresponds to 13 epochs for stage (Comb) and 9 epochs for stage (Sep). Stage (ML) was trained for 20 epochs which took 10h. Pre-training BERT took on average 25% longer. Fine-tuning used a batch size of 32 examples per device, 200 warm-up steps with a learning rate of 2e-05 which are the values reported by (Reusch et al., 2021). The fine-tuning of ALBERT took 8h for 3 epochs, while BERT needed 11h. Pre-training and fine-tuning was performed using Huggingface's library transformers (Wolf et al., 2020). To train and evaluate the structural probes, we used the original code provided by Hewitt et al. and adapted it to the transformers library[3]. We used the L1 loss and a maximum rank of the probe of 768 as reported by the authors.

6 Evaluation

6.1 Extending the tokenizer

First, we analyze whether it is necessary to add LaTeX tokens to the sub-word tokenizer. Hereby, we compare the results of six ALBERT models. The results of our AR fine-tuning task are shown in Tab. 1. Each model started from the ALBERT base checkpoint and was pre-trained using the indicated

[3]https://github.com/john-hewitt/structural-probes

	nDCG'	Both	Math	Text
ALBERT$^{\neg\mathrm{M}}$	0.365	0.369	0.347	0.403
ALBERT$^{\mathrm{M}}$	0.369	0.362	0.369	0.390
Sep$^{\neg\mathrm{M}}$	0.408	0.408	0.403	0.423
Sep$^{\mathrm{M}}$	0.413	0.399	0.417	0.440
ML+Sep$^{\neg\mathrm{M}}$	0.407	0.415	0.397	0.408
ML+Sep$^{\mathrm{M}}$	0.406	0.403	0.409	0.403
Comb$^{\neg\mathrm{M}}$*	0.401	0.411	0.389	0.404
Comb$^{\mathrm{M}}$	0.412	0.420	0.403	0.414
ML+Comb$^{\neg\mathrm{M}}$	0.396	0.402	0.389	0.397
ML+Comb$^{\mathrm{M}}$	0.403	0.404	0.411	0.381
μ^2ALBERT$^{\neg\mathrm{M}}$	0.409	0.408	0.407	0.416
μ^2ALBERT	0.427	0.423	0.428	0.432

Table 1: AR Results of six ALBERT models, $^{\neg\mathrm{M}}$ denotes the standard tokenizer, $^{\mathrm{M}}$ denotes the added LaTeX tokens, note that μ^2ALBERT implies that the $^{\mathrm{M}}$ was applied, hence it is not added to avoid confusion. * denotes the former state-of-the-art model using the same fine-tuning scheme.

stage and tokenizer. After adding the LaTeX tokens, almost all models show at least a small improvement in the nDCG' and the Math scores. For the categories "Both" and "Text", the results vary by model. The overall improvement on nDCG' when adding LaTeX is 0.007. Especially the nDCG' score on topics that require math understanding is improved by 0.017 points on average. For topics that depend on understanding "Text" the improvements are small: 0.001. Over the six models the average improvement for the category mathematics and text ("Both") was 0. Hence, we can conclude that adding LaTeX mainly benefits the model's math understanding and should be considered when dealing with mathematical notation, since it requires almost no effort. The largest overall improvement demonstrates μ^2ALBERT, while model (ML)+(Comb)$^{\mathrm{M}}$ improved the most in the Math category.

6.2 Ablation Study

Each of the stages we have presented improves the capacities of our models. We show this empirically by removing each of the stages from the pre-training of our ALBERT model resulting in total in four models: μ^2ALBERT, μ^2ALBERT without (ML), μ^2ALBERT without (Sep) and μ^2ALBERT without (Comb). The results on the AR fine-tuning task are summarized in Tab. 2. Regarding the met-

	nDCG'	mAP'	p@10'	Both nDCG'	Math nDCG'	Text nDCG'
μ^2ALBERT	**0.427**	**0.249**	0.341	**0.423**	**0.428**	**0.432**
Comb+Comb	0.401	0.222	0.332	0.389	0.415	0.394
Comb+Comb+Comb	0.399	0.224	0.329	0.384	0.417	0.398
μ^2ALBERT - ML	0.417	0.237	**0.357**	0.418	0.419	0.411
μ^2ALBERT - Comb	0.406	0.229	0.341	0.403	0.409	0.403
μ^2ALBERT - Sep	0.403	0.222	0.321	0.404	0.411	0.381
BERT base$^{\mathrm{M}}$	0.346	0.161	0.260	0.356	0.333	0.353
μ^2BERT	0.406	0.237	0.347	0.402	0.407	0.412

Table 2: AR results of μ^2ALBERT and μ^2BERT. Our models perform best across almost all metrics.

rics nDCG' and mAP', the full-model μ^2ALBERT with all stages performs best. The second best model was trained on two additional stages, using only the NL-LaTeX-separated and the combined data. It was not trained on the mathematics-only data. This model performed best on p@10', but in the long run its results were worse compared to the full model resulting in lower scores for nDCG' and mAP' which take the model's performance after more than the first ten results into account. When looking at the results per category, adding each stage improved all scores. Especially stage (Sep), the separation of NL and LaTeX, led to an improvement of 0.059 on nDCG' in the category 'Text'. The smallest impact overall had stage (ML), the LaTeX stage. Furthermore, we can say that our model's final performance it not simply a result of a longer pre-training. The models (Comb)+(Comb) and (Comb)+(Comb)+(Comb) were trained using two and three times the combined stage and showed signs of over-fitting after the second combined stage since the performance became slightly lower.

6.3 ALBERT vs. BERT

To save compute time, we performed the experiments so far using the ALBERT architecture. Nevertheless, BERT is still the more popular model. Therefore, we also applied the same training scheme on a pre-trained BERT checkpoint, resulting in μ^2BERT, and compared it to BERT base with additional LaTeX tokens added to the tokenizer. The results are displayed in Tab. 2 and show significant improvements on all reported metrics. μ^2BERT performed especially better over the previous state-of-the-art model (Comb)$^{\neg\mathrm{M}}$(see Tab. 1) in the 'Math' category with 0.018 points in nDCG'. In comparison to μ^2ALBERT, the scores are not as high, which can be expected since the pre-training scheme was initially optimized for ALBERT and it's SOP task.

Model	Best UUAS	Best DSpr.
SciBERT$^{\neg\mathrm{M}}$	0.5460	0.7267
BERT base$^{\neg\mathrm{M}}$	0.5327	0.7139
BERT base$^{\mathrm{M}}$	0.6327	0.7794
ALBERT$^{\mathrm{M}}$	0.4923	0.6572
μ^2ALBERT	0.5125	0.6800
μ^2BERT	**0.6998**	**0.8157**

Table 3: Results of reconstruction of OPTs using UUAS and DSpr., displaying only the best results across all 11 layers

6.4 Learned Mathematical Knowledge

After we have evaluated both models on the AR fine-tune task, which is an extrinsic evaluation, we now investigate how much mathematical knowledge was accumulated during training. Hereby, a structural probe was trained on top of the contextualized embeddings to reconstruct operator trees given a formula in LaTeX as input. We compare our models μ^2ALBERT and μ^2BERT to their base models as well as to SciBERT (Beltagy et al., 2019) since it is designed to model scientific documents which by nature include formulas and mathematical notation. We did not add additional LaTeX tokens to SciBERT and applied its official tokenizer. Fig. 3 shows the results of the probes trained on the embeddings of each of the models' layers, while Tab. 3 summarizes the highest values from all layers. We report our results using UUAS and DSpr. where higher values indicate a larger percentage of correctly reconstructed edges and a higher correlation between the predicted and gold-distances, respectively. First, we can see that BERT and SciBERT perform in a similar way. Surprisingly, both

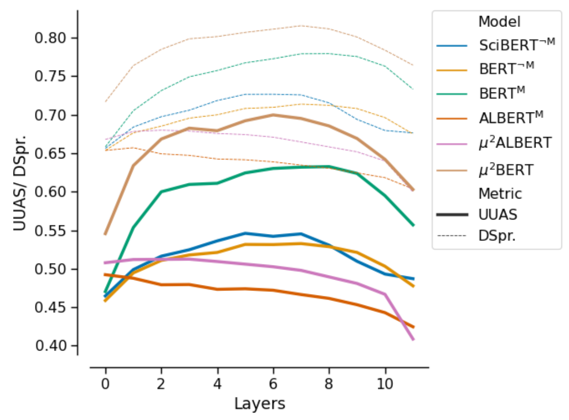


Figure 3: Results of reconstruction of OPTs across all model layers.

models already demonstrate a higher ability of extracting and reconstructing the OPTs from their embeddings compared to ALBERT$^{\mathrm{M}}$ and our trained μ^2ALBERT. By adding mathematical tokens to the tokenizer, the probe showed higher results in reconstructing OPTs which is visible from the comparison between BERT base$^{\neg\mathrm{M}}$ and BERT base$^{\mathrm{M}}$. The best result yields μ^2BERT while μ^2ALBERT produces even lower results than the BERT models without special support for mathematical tokens nor in-domain pre-training. As Fig. 3 shows, there exists a clear trend that most knowledge can be extracted from layers in the middle of the BERT-based models which is in line with the findings by Hewitt et al. who found the same pattern for BERT base and BERT large. Interestingly, we could not confirm this behavior for ALBERT. Here, the scores of all evaluated models decline with rising numbers of layers. We hypothesize that this might be caused by the weight sharing mechanism between layers in ALBERT. Examples of reconstructed OPT for μ^2BERT and μ^2ALBERT, as well as BERT$^{\mathrm{M}}$ and ALBERT$^{\mathrm{M}}$ are displayed in Fig. 4. In a large majority of cases the models correctly identified the edges of the displayed formula. Most differences can be seen from the second part of the left hand side of the equation. μ^2BERT reconstructed the OPT correctly, while the other three models mostly struggle with the child relationships of the equal sign.

7 Conclusion

Domain-adaptive pre-training has been proven to be effective even when adapting models for domains where coherent text is discontinued such as

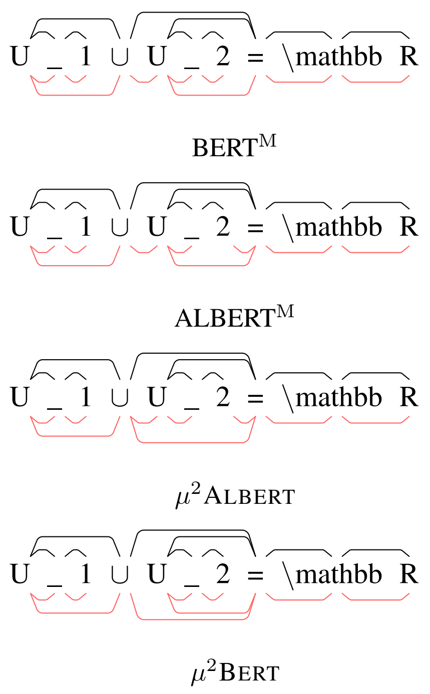

U _ 1 ∪ U _ 2 = \mathbb R
BERTM
U _ 1 ∪ U _ 2 = \mathbb R
ALBERTM
U _ 1 ∪ U _ 2 = \mathbb R
μ2ALBERT
U _ 1 ∪ U _ 2 = \mathbb R
μ2BERT

Figure 4: Operator Trees calculated from the predicted squared distances between the tokens. The black edges above each formula are the gold edges from the OPT parser, while the red edges are the predicted ones by each model.

programming code (Feng et al., 2020). In this work, we have introduced a pre-training scheme involving three different stages for language modeling of documents containing mathematical notation. We have demonstrated the usefulness of each stage and also of the decision to include LaTeX tokens for mathematical notation by applying the models to the task of mathematical answer ranking. Here, the models showed a better performance especially on questions involving mathematical understanding. Compared to the previous state-of-the-art model, μ^2ALBERT demonstrated improvements of 6.5% on the overall nDCG' score. In addition, we trained a structural probe on top of the models' contextualized embeddings to demonstrate that operator trees can be extracted. Here, only μ^2BERT showed strong results suggesting that it actually learned mathematical knowledge. For ALBERT this implication still remains unclear. Furthermore, whether the models' actually use the learned knowledge during downstream tasks is also unknown due to the nature of the applied probe. Further research is required to investigate these issues. Our models and the code are published in the project's Repository[4].

[4]Links anonymized for review

References

Emily Alsentzer, John R Murphy, Willie Boag, Wei-Hung Weng, Di Jin, Tristan Naumann, WA Redmond, and Matthew BA McDermott. 2019. Publicly available clinical bert embeddings. *NAACL HLT 2019*, page 72.

Iz Beltagy, Kyle Lo, and Arman Cohan. 2019. Scibert: A pretrained language model for scientific text. 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 3606–3611.

Cheng-Han Chiang, Sung-Feng Huang, and Hung-yi Lee. 2020. Pretrained language model embryology: The birth of albert. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6813–6828.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. 2019. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*.

Andy Coenen, Emily Reif, Ann Yuan, Been Kim, Adam Pearce, Fernanda Viégas, and Martin Wattenberg. 2019. Visualizing and measuring the geometry of bert. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 8594–8603.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

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

Rowan Hall Maudslay, Josef Valvoda, Tiago Pimentel, Adina Williams, and Ryan Cotterell. 2020. A tale of a probe and a parser. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7389–7395, Online. Association for Computational Linguistics.

John Hewitt and Christopher D Manning. 2019. A structural probe for finding syntax in word representations. 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 4129–4138.

Kexin Huang, Jaan Altosaar, and Rajesh Ranganath. 2019. Clinicalbert: Modeling clinical notes and predicting hospital readmission. *arXiv preprint arXiv:1904.05342*.

Hwiyeol Jo, Dongyeop Kang, Andrew Head, and Marti A Hearst. 2021. Modeling mathematical notation semantics in academic papers. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3102–3115.

Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2020. Learning and evaluating contextual embedding of source code. In *International Conference on Machine Learning*, pages 5110–5121. PMLR.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2020. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240.

Behrooz Mansouri, Douglas W Oard, and Richard Zanibbi. 2021. Dprl systems in the clef 2021 arqmath lab: Sentence-bert for answer retrieval, learning-to-rank for formula retrieval.

Behrooz Mansouri, Shaurya Rohatgi, Douglas W Oard, Jian Wu, C Lee Giles, and Richard Zanibbi. 2019. Tangent-cft: An embedding model for mathematical formulas. In *Proceedings of the 2019 ACM SIGIR international conference on theory of information retrieval*, pages 11–18.

Vít Novotnỳ, Petr Sojka, Michal Štefánik, and Dávid Lupták. 2020. Three is better than one. In *CEUR Workshop Proceedings. Thessaloniki, Greece*.

Vít Novotnỳ, Michal Štefánik, Dávid Lupták, Martin Geletka, Petr Zelina, and Petr Sojka. 2021. Ensembling ten math information retrieval systems.

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

Anja Reusch, Maik Thiele, and Wolfgang Lehner. 2021. Tu_dbs in the arqmath lab 2021, clef. In *CEUR Workshop Proceedings*.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2020. A primer in bertology: What we know about how bert works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

Shaurya Rohatgi, Jian Wu, and C Lee Giles. 2020. Psu at clef-2020 arqmath track: Unsupervised re-ranking using pretraining. In *CEUR Workshop Proceedings. Thessaloniki, Greece*.

Shaurya Rohatgi, Jian Wu, and C Lee Giles. 2021. Ranked list fusion and re-ranking with pre-trained transformers for arqmath lab.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R Bowman, Dipanjan Das, et al. 2019. What do you learn from context? probing for sentence structure in contextualized word representations. *arXiv preprint arXiv:1905.06316*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*, 30:5998–6008.

Jesse Vig. 2019. A multiscale visualization of attention in the transformer model. *arXiv preprint arXiv:1906.05714*.

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.

Wei Zhong, Shaurya Rohatgi, Jian Wu, C Lee Giles, and Richard Zanibbi. 2020. Accelerating substructure similarity search for formula retrieval. *Advances in Information Retrieval*, 12035:714.

Wei Zhong and Richard Zanibbi. 2019. Structural similarity search for formulas using leaf-root paths in operator subtrees. In *European Conference on Information Retrieval*, pages 116–129. Springer.