Injecting Numerical Reasoning Skills into Language Models

Mor Geva[*]
Tel Aviv University,
Allen Institute for AI
morgeva@mail.tau.ac.il

Ankit Gupta[*]
Tel Aviv University
ankitgupta.iitkanpur@gmail.com

Jonathan Berant
Tel Aviv University,
Allen Institute for AI
joberant@cs.tau.ac.il

Abstract

Large pre-trained language models (LMs) are known to encode substantial amounts of linguistic information. However, high-level reasoning skills, such as numerical reasoning, are difficult to learn from a language-modeling objective only. Consequently, existing models for numerical reasoning have used specialized architectures with limited flexibility. In this work, we show that numerical reasoning is amenable to automatic data generation, and thus one can inject this skill into pre-trained LMs, by generating large amounts of data, and training in a multi-task setup. We show that pre-training our model, GENBERT, on this data, dramatically improves performance on DROP ($49.3 \rightarrow 72.3$ F_1), reaching performance that matches state-of-the-art models of comparable size, while using a simple and general-purpose encoder-decoder architecture. Moreover, GENBERT generalizes well to math word problem datasets, while maintaining high performance on standard RC tasks. Our approach provides a general recipe for injecting skills into large pre-trained LMs, whenever the skill is amenable to automatic data augmentation.

1 Introduction

Recently, models trained on large amounts of data with a language modeling (LM) objective, have shown great promise in natural language processing, exhibiting surprising amounts of knowledge and information (Peters et al., 2018; Devlin et al., 2019; Liu et al., 2019; Lan et al., 2019; Petroni et al., 2019; Hewitt and Manning, 2019). However, high-level skills, such as the ability to perform numerical reasoning over text, can be challenging to capture with a LM objective only. Consider the example in Table 1. To solve the first question (Q1), a model must capture the value of numbers in the

---

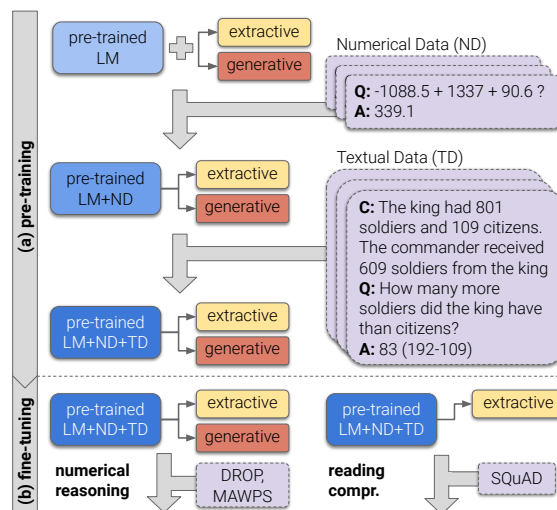[*]These authors contributed equally.

Figure 1: An overview of our approach for injecting numerical skills into a pre-trained LM. (a) We add two pre-training steps over large amounts of synthetic numerical data (ND) and textual data (TD); (b) we further fine-tune the model over either numerical reasoning datasets (DROP, MAWPS) or reading comprehension datasets (SQUAD).

text, compute their difference, and generate the tokens corresponding to the result, which generally do not appear in the input passage.

To make the task more manageable, state-of-the-art models have employed specialized architectures, restricting the space of possible numerical computations to a limited set. Modules were designed for counting (but only until '9') and for addition and subtraction (but of 2-3 numbers only). Such models perform well on existing datasets, such as DROP (Dua et al., 2019), but do not generalize to unsupported computations, which will require modifying the model architecture. Moreover, current models marginalize at training time over all numerical expressions that evaluate to the correct answer. Since the number of such expressions grows exponentially, scaling these approaches to arbitrary computations entails using non-differentiable operations (sampling or computing top-K numerical expressions), which can lead to training difficulties.

946

Passage: *Taunton has four art galleries... Hughes/ Donahue Gallery founded in 2007, a local community gallery serving local Taunton artists... Art Euphoric founded in 2008 has both visual and craft exhibits...*
Q1: How many years after founding of Hughes/ Donahue was Art Euphoric founded?
A1: 1 (number)
Q2: Which gallery was founded later, Hughes/ Donahue or Art Euphoric?
A2: Art Euphoric (span)

Table 1: Example passage from DROP, and two questions with different answer types.

In this work, we propose that reasoning skills, such as numerical reasoning, are amenable to *automatic data generation*. Hence, one can inject that skill directly into the model by adding additional pre-training steps, allowing the model to learn the skill in an end-to-end fashion. This results in a fully-differentiable training procedure over a standard and general-purpose architecture, where the output space can be easily controlled through the data generation procedure.

Specifically (Figure 1), we add to a large pre-trained LM two pre-training steps over automatically-generated synthetic data. First, we generate *numerical data* of the form $3 + 4 + 11 = 18$. Training on this data teaches the model to compute the value of numbers from their tokens and to perform numerical operations. Second, we automatically generate question-passage pairs that require numerical reasoning using a compact grammar (*textual data*). Training on this data endows the model with the ability to understand computations expressed in pseudo-natural language.

In both pre-training steps, the model, GEN-BERT, generates output numbers token-by-token. Thus, the model has a standard architecture, where an answer can either be extracted from the input question and passage or generated from a decoder. Pre-training is done in a multi-task setup with a standard LM objective, in order to avoid "catastrophic forgetting" (Kirkpatrick et al., 2017) of the linguistic information in the original LM. After pre-training, the model has sufficient language and numerical skills to be directly fine-tuned on a target numerical reasoning dataset, without resorting to specialized architectures. Augmenting more numerical skills does not require changing the model, only generating additional data.

We demonstrate the validity of our approach by a series of experiments showing that:

(a) GENBERT is able to solve pre-training tasks for numerical reasoning.

(b) Pre-training on these tasks provides GEN-BERT with 1) skills to reach performance that matches state-of-the-art models of comparable size on DROP (Dua et al., 2019), a standard numerical reasoning dataset, as well as 2) the ability to generalize to math word problem (MWP) datasets (Koncel-Kedziorski et al., 2016).

(c) GENBERT learns these numerical skills while maintaining high performance on SQuAD (Rajpurkar et al., 2016), a standard reading comprehension dataset.

(d) Initializing models for numerical reasoning with GENBERT's weights improves their original performance.

To conclude, in this work we address the problem of injecting LMs with numerical reasoning skills. Our contributions are:

- A method for injecting skills into pre-trained LMs, given that automatic data generation is possible.
- GENBERT, an architecture for pre-trained LM with generative and extractive abilities.
- A framework for generating numerical and textual synthetic data for numerical reasoning.

Our code and data can be downloaded from https://github.com/ag1988/injecting_numeracy.

2 Numerical Reasoning Over Text

Numerical reasoning over text (NRoT) is commonly set up as a reading comprehension (RC) task. Given a training set of question-context-answer triples $\{(\mathbf{q}_i, \mathbf{c}_i, a_i)\}_{i=1}^N$, the goal is to learn a function that returns the answer a to a question $\mathbf{q}$ given a context $\mathbf{c}$. However, in NRoT the answer generally requires to internally perform some numerical computation using the entities and numbers in the context. Specifically, the answer is either: (a) a span (or list of spans) from the context $\mathbf{c}$ or question $\mathbf{q}$, or (b) a number that is the result of some computation (see examples in Table 1).

Two natural, yet opposing, approaches lend themselves to tackling NRoT: (a) *A symbolic approach*: a model can read the question and context, output a numerical expression and evaluate the answer with an external symbolic calculator. This approach is a particular case of semantic parsing (Kamath and Das, 2019), and was common in early NRoT datasets (Koncel-Kedziorski et al., 2015; Roy and Roth, 2015; Hosseini et al., 2014). How-

ever, it suffers from several drawbacks. First, because numerical expressions are discrete and their space grows combinatorially, the model must learn to search in this space using non-differentiable operations, which are usually difficult to optimize. Second, numerical expressions are limited to numerical answers, while in DROP often a numerical computation is required but the final answer is a text span. (b) *A distributed approach*: have a model directly generate the answer given $(\mathbf{q}, \mathbf{c})$. When the answer is a text span, the model can extract it from the input, and when the answer is a number that is not in $\mathbf{q}$ or $\mathbf{c}$, the model must generate it. While this makes training straightforward, the model must learn to perform numerical computations from the relatively small target dataset. We empirically show in §3 that this leads to low performance in general.

As a compromise, most NRoT models (Dua et al., 2019; Kinley and Lin, 2019; Hu et al., 2019; Efrat et al., 2019) have taken a hybrid approach: they augment standard extractive QA models with specialized modules for handling a limited set of numerical computations. We briefly describe this architecture, as it is the basis for our model in §3.

Given a question with n_1 tokens $\mathbf{q} = (q_1, \ldots, q_{n_1})$ and a context with n_2 tokens $\mathbf{c} = (c_1, \ldots, c_{n_2})$, the hybrid model first computes contextualized representations for the $n_1 + n_2 + 3$ tokens $\langle$[CLS] $\mathbf{q}$ [SEP] $\mathbf{c}$[SEP]$\rangle$ using a pre-trained LM, such as BERT (Devlin et al., 2019):

$$\mathbf{L} = \mathbf{LM}(\mathbf{q}, \mathbf{c}).$$

The representations $\mathbf{L}$ are then passed to multiple *heads*, which are small neural networks that estimate $p(a \mid \mathbf{q}, \mathbf{c}, h)$, that is, the probability of the answer given the input and conditioned on a head h, corresponding to a particular answer type:

- *Context span head*: computes a distribution over all spans in the *context* using a feed-forward network (FFN) $\mathbf{FF_c}(\mathbf{L})$.
- *Question span head*: computes a distribution over spans in the *question* using a FFN $\mathbf{FF_q}(\mathbf{L})$.
- *Count head*: computes a distribution over the numbers $\{0, \ldots, 9\}$ using a FFN $\mathbf{FF_{cnt}}(\mathbf{L})$.
- *Arithmetic head*: computes a distribution over all signed combinations of numbers in the context using a FFN $\mathbf{FF_{cmb}}(\mathbf{L})$ (the numbers in the context are identified in a pre-processing step).

While the first two heads are standard in extractive QA, the latter two heads are specialized and meant to handle answers that do not appear in the input.

Finally, for deciding which answer head to use for a given input, a *type* head $\mathbf{FF_{typ}}(\mathbf{L})$ outputs a probability distribution $p_{\text{head}}(h \mid \mathbf{q}, \mathbf{c})$ (using a FFN). Thus the model probability for an answer is

$$p(a \mid \mathbf{q}, \mathbf{c}) = \sum_{h \in \text{heads}} p_{\text{head}}(\mathbf{h} \mid \mathbf{c}, \mathbf{q}) \cdot p(a \mid \mathbf{c}, \mathbf{q}, h).$$

Training is done by enumerating all of the ways in which the answer can be obtained using all of the heads, and maximizing this marginal probability.

While existing models perform well on DROP, the aforementioned architecture is not flexible. First, the output space is severely constrained – the model can only count up to '9', and numerical computations are restricted to signed combinations of a few numbers. Second, expanding the space of supported numerical computations is non-trivial, because training involves marginalizing over all expressions that lead to the correct answer. Since the space of numerical expressions grows exponentially, expanding this space quickly leads to a difficult search problem. Third, delegating numerical computations to an external symbolic calculator leads to modeling challenges, since there could be interactions between text and numerical computation: Consider the DROP question *"How many total yards did Phil Dawson throw for touchdowns?"*. Current models handle such questions by computing a sum from numbers in the text and returning the result. However, if the question was *"Who threw 45 total yards for touchdowns?"*, the model would have to compute the sum *internally*, and then find the relevant span in the text. This is impossible when the computation itself is delegated to an external calculator. Thus, training models to handle such numerical questions is desirable.

Motivated by the above arguments, we wish to push the frontier of end-to-end differentiable models for numerical reasoning. Thus, we will automatically generate large amounts of data that endow a pre-trained LM with numerical skills.

3 GENBERT: A BERT-based Model for Generating Arbitrary Outputs

We now describe a simple BERT-based generative model that performs numerical computations internally, termed GENBERT. The model combines the Transformer encoder-decoder architecture (Vaswani et al., 2017) with a pre-trained LM, specifically, BERT.

Our architecture is illustrated in Figure 2. Our encoder is a standard Transformer, initialized with

948

BERT weights. To also enjoy BERT's represen-tations at decoding time, we tie the weights of the decoder and the encoder. Because the Transformer decoder has *source attention* weights (weights for attending to the encoder representations at decod-ing time) that are not present in BERT, we tie these source-attention weights to the self-attention weights of the encoder (which are tied to the self-attention weights of the decoder). This fully initial-izes the Transformer model with BERT weights.

Since the encoder and decoder weights are tied, we make them learn distinct representations by adding a FFN $\mathbf{FF_{enc}}$ that transforms the encoder contextualized representations $\mathbf{L_{enc}}$ as

$$\mathbf{H_{enc}} = \texttt{layer-norm}(\texttt{gelu}(W \cdot \mathbf{L_{enc}})),$$

where W is a parameter matrix (Hendrycks and Gimpel, 2016; Ba et al., 2016). Analogously, we add $\mathbf{FF_{dec}}$ to the decoder. To further dis-tinguish the encoder and decoder, we use dis-tinct start and end tokens for input and out-put sequences. Given m answer tokens $\mathbf{a} = (a_1, \ldots, a_m)$, we form an output sequence with $m + 2$ tokens: $\langle \texttt{[SOS]}\ \mathbf{a}\ \texttt{[EOS]} \rangle$. The output tokens are passed through the decoder and $\mathbf{FF_{dec}}$ to obtain $\mathbf{H_{dec}}$.

Finally, the probability of an answer is defined in the usual manner: Let $\langle \mathbf{a} \rangle = (a_0 \cdots a_{m+1})$ be the output sequence. The decoder outputs the probabil-ity $p_{\text{dec}}(a_{i+1} \mid a_0, ..a_i, \mathbf{c}, \mathbf{q})$, and the probability of an answer is:

$$p_{\text{dec}}(\langle \mathbf{a} \rangle \mid \mathbf{c}, \mathbf{q}) = \prod_{i=0}^{m} p_{\text{dec}}(a_{i+1} \mid a_0, ..a_i, \mathbf{c}, \mathbf{q}).$$

As we have a generative model, we can remove the specialized *count* and *arithmetic* heads from §2. Thus, the type head $\mathbf{FF_{typ}}(\mathbf{H_{enc}})$ outputs a distribution $(p_\mathbf{q}, p_\mathbf{c}, p_{\text{dec}})$ over the context span, question span, and decoder heads.

To improve pre-training on the numeric data (§4), we make two additional modifications.

Digit Tokenization (DT) Conventional word-piece tokenization treats numbers no differently than any other token. However, computing the value of numbers should be simpler when using digits directly (Wallace et al., 2019). Hence, we tokenize numbers digit-by-digit. For example, a wordpiece $\texttt{\#\#}d_1 \cdots d_k$ where $d_i \in \{\texttt{0}, \ldots, \texttt{9}\}$ is further split into $\texttt{\#\#}d_1, ..., \texttt{\#\#}d_k$. We show in §5.1 that this substantially improves sample complexity when training to perform numerical operations.
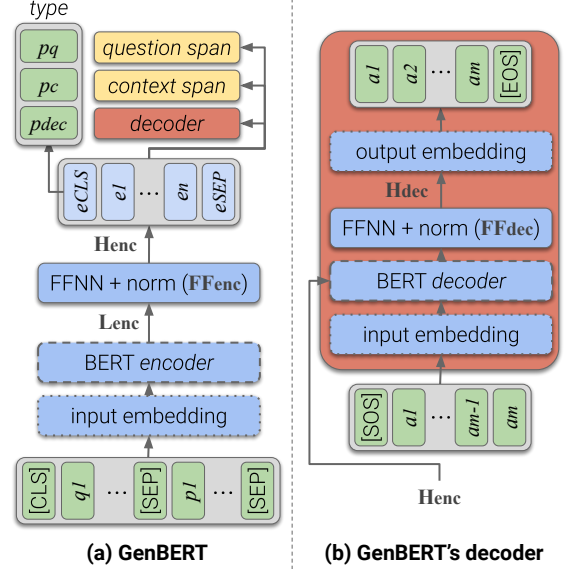

Figure 2: GENBERT's network architecture: (a) a high-level overview of the network, including a generative head (red), two span-extraction heads (yellow), and an answer type head. (b) a closer overview of GENBERT's generative head.

Random Shift (RS) The original Transformer uses absolute positional embeddings for each to-ken. However, in §4, we train on short inputs such as *"1086.1 - 2.54 + 343.8"*. Thus, the model can potentially over-fit and learn to perform numerical reasoning only when numbers are at the beginning of an input. To prevent this, when the input length $n_1 + n_2 + 3 < 512$, we shift all position IDs by a random integer in $(0, 1, \ldots, 512 - (n_1 + n_2 + 3))$.

Training For each span (i, j), a span extraction head h outputs its probability $p_h((i, j) \mid \mathbf{c}, \mathbf{q}, h)$ of being the answer. Let S be the set of spans in the input corresponding to the gold answer. The model loss $\mathbf{L}_{\text{model}}$ marginalizes over all ways in which the answer can be predicted:

$$-\log \left(p_{\mathbf{dec}} \cdot p_{\text{dec}}(\langle \mathbf{a} \rangle) + \sum_{\mathbf{h} \in \mathbf{q}, \mathbf{c}} p_\mathbf{h} \cdot \sum_{(i,j) \in S} p_h(i, j) \right),$$

where conditionals have been dropped for brevity.

To evaluate the ability of GENBERT to perform numerical reasoning, we initialize it with BERT and fine-tune it on DROP. GENBERT obtains 46.1 EM and 49.3 F_1, roughly 20 points lower than prior models. Thus, we conclude that acquiring numerical reasoning skills from DROP data only is difficult. To remedy this, we will automatically generate training data that will endow GENBERT with numerical skills before training it on DROP.
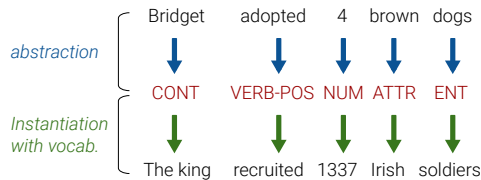
Figure 3: Template extraction and instantiation. A template (in red) is extracted from a MWP sentence, using categories for containers, entities, verbs, attributes and numbers, according to Hosseini et al. (2014). For generation, the categories are instantiated with a domain-specific vocabulary.

4 Pre-training Tasks for Numerical Skills

We now describe two automatically-generated datasets and the multi-task training procedure.

4.1 Generating Numerical Data (ND)

Our first dataset focuses on learning numerical values expressed by tokens and computing numerical operations, i.e., it does not involve textual content. As such, it is easy to craft templates that correspond to various numeric operations. We designed six such templates, described in Table 2. Each template consists of an expression to evaluate and its solution. Further details on their instantiation are provided in §A.1. While the numerical operations were chosen based on DROP, it is trivial to extend them to other domains (Saxton et al., 2019) with different numerical operations.

4.2 Generating Textual Data (TD)

Numeric data is easy to generate, since it does not contain any textual context. However, to tackle NRoT, a model needs to comprehend how numerical operations are expressed in text that refers to events, entities and quantities. This primes us to generate *textual data* from a simple grammar.

While text generation is hard in the general case, we are specifically interested in text that focuses on number manipulations. Therefore, we use the framework of Hosseini et al. (2014), who proposed to model math word problems with a simple structure. In their framework a *world state* consists of *entities*, which are objects that are being counted, and *containers*, which are objects that own entities. Sentences use *verb categories* to describe how the number of entities in a container changes, and thus a world state can be updated given a sentence.

Consider the textual example in Figure 1. the *entities* are soldiers and citizens, and the *containers* are the king and the commander. The verbs (*"had"* and *"received"*) describe the entities the king holds, and how many were passed to the commander.

In this work, we use this framework to automatically generate examples. We *extract templates* that describe changes in the number of entities owned by containers, and automatically *generate question-context pairs* from these templates.

Template extraction To extract templates, we go over sentences from the corpus provided by Hosseini et al. (2014). For each sentence, we use a procedure described by Hosseini et al. (2014) to abstract its tokens to the following categories: numbers (NUM), entities (ENT), containers (CONT) and attributes (ATTR). In addition, verbs are abstracted to six categories, each corresponding to a different change in the number of entities owned by containers. Thus, each template fully specifies how to update a *world state*, i.e., the number of entities each container owns. The top part of Figure 3 illustrates the abstraction process. Finally, we count for each extracted template its frequency in the data, and use the top-12 templates for passage generation. Details on the abstraction process, categories used, and extracted templates are in §A.2.

Passage generation Using the extracted templates, we can generate sentences and maintain a world state of all containers and the number of entities they own. We construct a small vocabulary (<100 words) that maps categories to domain-specific words, and use the following procedure to generate passages.

We sample 3-6 templates with replacement, and instantiate them one-by-one (the bottom part of Figure 3 illustrates instantiation). Each template is instantiated by uniformly sampling values from the vocabulary with probability $1 - p$ and from previously generated sentences with probability p. To avoid a collection of unrelated sentences, we set the probability of using previously used values to $p = 0.7$. An example passage is shown in Table 3.

Question generation After generating a passage, the world state holds information about all containers in the passage and the number of entities they hold. In Table 3, the state will include the number of families and rebels of different nationalities in each container (the commander, the householder, and the countries). Based on this world state, numerical reasoning questions can be asked.

To create questions, we craft 13 question templates that are instantiated with objects from the world state. The questions teach the model to track events and perform numeric and discrete operations.

Operation	Template	Example instantiation
signed float combination	$s_1\ f_1\ s_2\ f_2\ s_3\ f_3\ s_4\ f_4$	517.4 - 17484 - 10071.75 + 1013.21
min/max/avg	$o(f_1,\ f_2,\ f_3,\ f_4)$	largest(13.42, 115.5, 72.76)
arg max, arg min	$arg(w_1\ f_1,\ w_2\ f_2,\ w_3\ f_3,\ w_4\ f_4)$	arg min(highish 137.1, sightliness 43.2)
date min/max	$dsup(d_1,\ d_2,\ d_3,\ d_4)$	oldest(June 04, 959; 01 May 959)
date difference	diff in $prd(d_1, d_2)$	diff in days(05 April 112; June 01, 112)
percentage	$pcent\ w\ ::\ w_1\ p_1\%,\ w_2\ p_2\%,\ w_3\ p_3\%,\ w_4\ p_4\%$	percent not sunbird :: sunbird 33.2%, defector 60.77%, molehill 6.03%

Table 2: Templates for generating synthetic numerical examples and the numerical operations required to answer them. **Domains** (defined in App. A.1): $s_i \in \{-, +\}$, $f_i \in \mathbb{R}^+$, $o \in \mathcal{O}$: superlative words like *"longest"*, $arg \in \{\arg\min, \arg\max\}$, $w_i \in \mathcal{W}$: words from NTLK Words Corpus, $d_i \in \mathcal{D}$: dates until Sep 2019, $dsup \in \mathcal{DSUP}$: superlative words like *"latest"*, $prd \in \{$*"days"*, *"months"*, *"years"*$\}$, $p_i \in (0, 100)$, $pcent \in \{$*"percent"*, *"percent not"*$\}$.

P: The commander recruited 1949 Polish families in Spain. The householder recruited 1996 Japanese families in Spain. There were 10913 white rebels and 77 Chinese families in Spain. 6641 British soldiers, 476 asian rebels, and 338 Germans families were recruited in Russia.

Q: How many Japanese families were in Spain?
A: 1996
Q: How many more Japanese families were in Spain than Polish families?
A: 47 (1996-1949)
Q: How many families of Spain were not Polish families?
A: 2073 (4022-1949)

Table 3: An example synthetic passage (P) and questions. Questions (Q) were generated from templates and answers (A) were calculated based on the world state.

Examples for generated questions are shown in Table 3, where answers are computed from the world state. Overall, we create 13 question templates for 7 different "skills", provided in §A.2.

4.3 Training GENBERT on Synthetic Data

For pre-training on ND, we generated 1M examples for training and 10K for validation. For TD, we generated 2.5M examples for training and 10K for validation. For both synthetic datasets, we used the GENBERT model loss, $\mathbf{L}_{\text{model}}$, from §3. To ensure that the model does not lose its language understanding abilities, we employ a multi-task setup, and include a standard *masked LM* objective from BERT. Specifically, given a masked token sequence $\langle \mathbf{m} \rangle$, we compute the contextualized representations, $\mathbf{L}_{\text{enc}}$ and pass them through a feed-forward network $\mathbf{FF}_{\text{mlm}}$. For each masked index i, it outputs the probability $p(a_i \mid i, \langle \mathbf{m} \rangle)$ of the original token a_i. The MLM loss is computed as

$$\mathbf{L}_{\text{mlm}}(\langle \mathbf{m} \rangle) = \text{mean}_{i \in \text{masked}} -\log(p(a_i \mid i, \langle \mathbf{m} \rangle)).$$

Details about the MLM data are in §A.3.

During training, we sample mini-batches from the respective datasets, and minimize the weighted sum of the losses. Concretely, while pre-training on ND and TD, we sample mini-batches X_{ND}, X_{TD} and X_{MLM} and optimize the objective

$$\mathbf{L}_{\text{model}}(X_{\text{ND}}) + \mathbf{L}_{\text{model}}(X_{\text{TD}}) + \lambda \cdot \mathbf{L}_{\text{mlm}}(X_{\text{MLM}}).$$
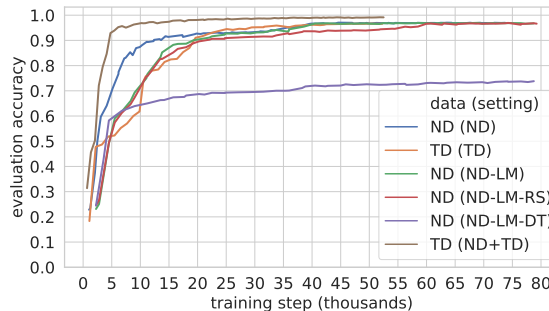

Figure 4: Progression of eval accuracy (EM) of GENBERT, for different pre-training settings listed in §5.1.

5 Experimental Evaluation

We now evaluate our two pre-training steps and their applicability for numerical reasoning tasks. We consider the following variants, aiming to investigate the contributions of ND and TD, the importance of MLM loss, and techniques like DT and RS. In all cases, we initialize GENBERT with BERT-base-uncased, use DT and RS, and include the MLM loss, *except where noted*:

- GENBERT$_{+\text{ND}}$: trained on numerical data.
- GENBERT$_{+\text{ND-LM}}$: trained on ND without the additional MLM loss.
- GENBERT$_{+\text{ND-LM-DT}}$: trained on ND using wordpiece tokenization, without the MLM loss.
- GENBERT$_{+\text{ND-LM-RS}}$: trained on ND without MLM loss and random shift (RS).
- GENBERT$_{+\text{TD}}$: trained on textual data (TD).
- GENBERT$_{+\text{ND+TD}}$: GENBERT$_{+\text{ND}}$ trained on both ND and TD.

5.1 Pre-training Performance

We first ask whether the pre-training procedure allows GENBERT to absorb the intended numerical skills. We observe that across various settings (ND, TD, ND+TD), GENBERT consistently achieves more than 96% accuracy in predicting the correct solution for both ND and TD. Thus, we conclude that indeed a pre-trained LM can learn the designed skills from generated data.

951

Figure 4 shows the learning curves of GEN-BERT for the different variants. Note that in ND-LM-DT the model does *not* learn to solve the numerical data task. This demonstrates the utility of using DT over conventional wordpieces. The lower sample complexity in the case of ND+TD compared to the only-TD can be attributed to the fact that ND and TD share some numeric skills and hence a model already trained on ND converges faster on TD compared to GENBERT.

5.2 Numerical Reasoning Performance

After successfully injecting GENBERT with numeric skills, we test GENBERT guided by the following questions:

(a) Are the injected skills robust and generalize to NRoT datasets like DROP?

(b) Are the new skills learned at the expense of the model's ability to understand language?

(c) Can the pre-trained weights be used with architectures other than GENBERT?

For (a), we fine-tune GENBERT on DROP and further evaluate on MWP in a zero-shot setup . For (b), we evaluate GENBERT on a RC task that does not involve numerical reasoning, namely, SQUAD (Rajpurkar et al., 2016). For (c), we use GENBERT encoder as a drop-in replacement for BERT on two other architectures.

Results on DROP We report results of GEN-BERT initialized by BERT-base and leave pre-training a larger model for future work. We compare GENBERT to MTMSN (Hu et al., 2019) initialized with BERT-base, as MTMSN initialized with BERT-large is a state-of-the-art model on DROP.[1]

Table 4 presents fine-tuning results on DROP. Without pre-training, GENBERT performs poorly compared to current state of the art models like MTMSN, reporting an EM of only 46.1. Pre-training on each of the numerical data (ND) and textual data (TD) improves performance dramatically to 64.7 EM and 64.4 EM, respectively. Moreover, pre-training on both ND and TD leads to a performance of 68.8 EM, on par with MTMSN's 68.2 EM. This demonstrates that the skills that GENBERT learns from ND and TD are complementary. In addition, the lower performance of GENBERT+ND-LM and GENBERT+ND-LM-RS

[1] Per ACL policy, we compare to models that were made public 3 months prior to submission.

	Development		Test	
	EM	F$_1$	EM	F$_1$
GENBERT	46.1	49.3	-	-
GENBERT+ND-LM-RS	61.5	65.4	-	-
GENBERT+ND-LM	63.8	67.2	-	-
GENBERT+ND	64.7	68.2	-	-
GENBERT+TD	64.4	67.8	-	-
GENBERT+ND+TD	**68.8**	72.3	**68.6**	**72.4**
NABERT+	63.0	66.0	61.6	65.1
MTMSN$_{BASE}$	68.2	**72.8**	-	-

Table 4: Performance of GENBERT and comparable models on the development and test sets of DROP.

	number	span	date	spans
GENBERT	42.3	67.3	47.5	21.1
GENBERT+ND	70.5	71.0	54.5	24.2
GENBERT+TD	69.2	72.6	55.2	22.0
GENBERT+ND+TD	**75.2**	**74.5**	**56.4**	24.2
NABERT+	67.8	69.2	39.8	22.4
MTMSN$_{BASE}$	**75.0**	71.3	44.2	**53.4**

Table 5: F$_1$ scores on DROP development per answer type.

shows the importance of including the MLM loss and the utility of RS for short inputs.

Breaking down performance by answer type (Table 5) highlights several points. First, pre-training on ND and TD improves performance mostly due to number answer types, as expected. Second, GENBERT+ND+TD outperforms MTMSN$_{BASE}$ on questions whose answer is *a span*. We argue a probable cause for this are span questions that require performing a numerical computation internally, as explained in §2. Third, MTMSN$_{BASE}$ substantially outperforms GENBERT on questions whose answer is a list of non-contiguous spans. This is expected, as MTMSN has a specialized head and procedure for handling such questions, while build on a simpler and more standard RC architecture.

Generalization to MWP (zero-shot) The MAWPS repository is a collection of math word problem (MWP) datasets (Koncel-Kedziorski et al., 2016). To test the models on skills they were trained on, we picked datasets with addition and subtraction problems, and filtered out examples with other operations (e.g., multiplication and division). All models that were fine-tuned on DROP were evaluated in a zero-shot setup on 395 examples from ADDSUB (Hosseini et al., 2014), 321 from SOP (Roy et al., 2015), and 305 from SEQ (Koncel-Kedziorski et al., 2015).

Results are shown in Table 6. Overall, GENBERT+ND+TD dramatically improves performance compared to GENBERT. GENBERT+ND performs much better than GENBERT+TD, demonstrating the utility of ND when the context is short.

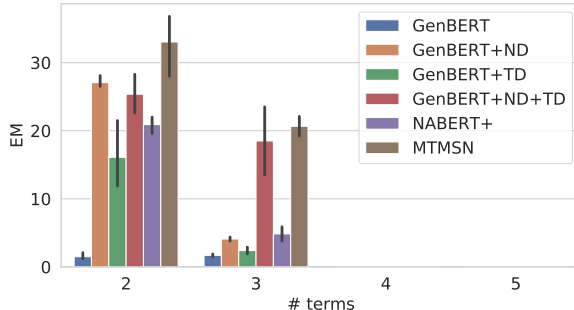|	ADDSUB	SOP	SEQ
GENBERT	2	1.2	1.3
GENBERT$_{+ND}$	22.8	26.5	23
GENBERT$_{+TD}$	10.4	21.5	12.1
GENBERT$_{+ND+TD}$	22.8	**28.3**	22.3
NABERT+	19.2	19.6	17.4
MTMSN$_{BASE}$	**32.2**	28	**32.5**

Table 6: EM on MWP datasets.

Figure 5: Breakdown of model accuracy (EM) by the number of terms in the arithmetic expression, for the MWP datasets ADDSUB, SOP and SEQ.

Last, MTMSN outperforms GENBERT$_{+ND+TD}$. However, MTMSN uses a specialized architecture for addition and subtraction, suitable when calculations are done outside of the model. GENBERT, on the other hand, is a general-purpose generative model, that can also return span answers when the computation is done internally.

Next, we break down performance by the number of terms in the arithmetic expression (Figure 5). The plot shows that all models struggle to generalize to more complex problems, and completely fail when the calculation involves more than 3 terms. Interestingly, the drop in performance of GENBERT$_{+ND+TD}$ between 2 and 3 terms is significantly smaller than that of GENBERT$_{+ND}$ and GENBERT$_{+TD}$. This suggests that both ND and TD are useful for improving robustness.

Error analysis To understand the limitations of our method, we analyze the errors of GENBERT$_{+ND+TD}$ on the development set of DROP, excluding questions with a multi-span answer which are not supported by the model. We sample 100 random examples for which GENBERT$_{+ND+TD}$ fails to predict the correct answer and manually analyze the types of questions and mistakes done by the model.

We find that in almost half of the cases (43%), the example requires reasoning skills that are either not covered by the pre-training tasks (e.g. sorting), or not numerical. Another common case (23%) is inaccurate predictions, such as spans that are too

	EM	F$_1$
BERT	**81.1**	**88.6**
GENBERT$_{+ND-LM}$	78.1	85.8
GENBERT$_{+ND}$	80.7	88.1
GENBERT$_{+TD}$	80.7	88.2
GENBERT$_{+ND+TD}$	**81.3**	**88.6**

Table 7: Performance on SQuAD v1 development set. Scores for BERT are using wordpiece tokenization.

long and numbers with partial digit match to the gold answer. We note that many of these errors can be addressed by extending the pre-training tasks to cover additional numerical skills and a larger number range. We leave such extensions for future work. Further details and example failure cases are provided in §A.5.

5.3 Reading Comprehension Performance

Having shown that our models successfully learned to perform NRoT, we investigate if this improvement comes at the expense of performance on RC datasets. We initialize the RC model from Devlin et al. (2019) with GENBERT weights (encoder only) and fine-tune it on SQUAD v1. As shown in Table 7, the performance of GENBERT$_{+ND+TD}$ is almost identical to the original BERT. Moreover, GENBERT$_{+ND-LM}$ reported a loss of 3 EM points highlighting the importance of using the MLM loss.

5.4 GENBERT With Other Architectures

To further establish the utility of GENBERT, we used the weights of GENBERT$_{+ND+TD}$ to initialize the encoder of NABERT+ and MS-TAG, a recent multi-span tagging model of Efrat et al. (2019). Fine-tuning on DROP shows an improvement of ∼2 EM points compared to the originally reported performance: 63.0 → 65.1 EM for NABERT+, and 67.3 → 69.3 EM for MS-TAG. This shows that GENBERT can be used as a drop-in replacement for BERT, when numerical reasoning is needed.

To summarize, we have empirically shown that one can inject numerical reasoning skills into a pre-trained LM, resulting in good performance on DROP, generalization to MWP, while maintaining high performance on standard RC datasets. Moreover, the resulting weights can be used for initializing numerical reasoning models.

6 Related Work

Most NRoT models designed for DROP are extractive QA models augmented with specialized modules (§2). Two recent work (Andor et al., 2019;

Chen et al., 2020) take a more symbolic approach and output a symbolic program augmented with operations over text. In our work, numerical computations are latent and performed internally by the model.

A related line of work has been analyzing the mathematical reasoning abilities of neural models over text (Wallace et al., 2019; Rozen et al., 2019; Ravichander et al., 2019), and on arithmetic problems (Saxton et al., 2019; Amini et al., 2019; Lample and Charton, 2020).

Designing pre-training tasks to teach LMs additional skills has been applied by Huang et al. (2019), who designed cross-lingual pre-training tasks to teach better mappings between languages, and Lee et al. (2019), who introduced the Inverse Cloze Task to pre-train an information retriever.

7 Conclusions

Large pre-trained LMs lack high-level skills such as numerical reasoning. Consequently, current models that perform numerical reasoning over a pre-trained LM resorted to customized modules with limited flexibility. In this work, we propose a general method for injecting additional skills into LMs, assuming automatic data generation is possible. We apply our approach to the task of numerical reasoning over text, using a general-purpose model called GENBERT, and a simple framework for generating large amounts of synthetic examples. Our experiments demonstrate the effectiveness of our method, showing that GENBERT successfully learns the numerical skills, and performs on par with state-of-the-art NRoT models of the same size.

Acknowledgments

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *North American Association for Computational Linguistics (NAACL)*, pages 2357–2367, Minneapolis, Minnesota.

Daniel Andor, Emily Pitler, Kenton Lee, and Luheng He. 2019. Giving bert a calculator: Finding operations and arguments with reading comprehension. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *North American Association for Computational Linguistics (NAACL)*, pages 4171–4186, Minneapolis, Minnesota.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *North American Association for Computational Linguistics (NAACL)*.

Avia Efrat, Elad Segal, and Mor Shoham. 2019. Tag-based multi-span extraction in reading comprehension. *arXiv preprint arXiv:1909.13375*.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

John Hewitt and Christopher D Manning. 2019. A structural probe for finding syntax in word representations. In *North American Association for Computational Linguistics (NAACL)*, pages 4129–4138.

M. J. Hosseini, H. Hajishirzi, O. Etzioni, and N. Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533.

Minghao Hu, Yuxing Peng, Zhen Huang, and Dongsheng Li. 2019. A multi-type multi-span network for reading comprehension that requires discrete reasoning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Haoyang Huang, Yaobo Liang, Nan Duan, Ming Gong, Linjun Shou, Daxin Jiang, and Ming Zhou. 2019. Unicoder: A universal language encoder by pre-training with multiple cross-lingual tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Aishwarya Kamath and Rajarshi Das. 2019. A survey on semantic parsing. Automated Knowledge Base Construction (AKBC).

Jambay Kinley and Raymond Lin. 2019. *NABERT+: Improving Numerical Reasoning in Reading Comprehension.*

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. In *Transactions of the Association for Computational Linguistics (TACL)*, volume 3, pages 585–597. MIT Press.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Association for Computational Linguistics (ACL)*, pages 1152–1157, San Diego, California.

Guillaume Lample and FranÃ§ois Charton. 2020. Deep learning for symbolic mathematics. In *International Conference on Learning Representations (ICLR)*.

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*.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *Association for Computational Linguistics (ACL)*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. 2018. Deep contextualized word representations. In *North American Association for Computational Linguistics (NAACL)*.

Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander Miller, and Sebastian Riedel. 2019. Language models as knowledge bases? In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Abhilasha Ravichander, Aakanksha Naik, Carolyn Rose, and Eduard Hovy. 2019. EQUATE: A benchmark evaluation framework for quantitative reasoning in natural language inference. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 349–361, Hong Kong, China. Association for Computational Linguistics.

S. Roy, T. Vieira, and D. Roth. 2015. Reasoning about quantities in natural language. *Transactions of the Association for Computational Linguistics (TACL)*, 1.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1743–1752.

Ohad Rozen, Vered Shwartz, Roee Aharoni, and Ido Dagan. 2019. Diversify your datasets: Analyzing generalization via controlled variance in adversarial datasets. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 196–205, Hong Kong, China. Association for Computational Linguistics.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations (ICLR)*.

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

Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. 2019. Do nlp models know numbers? probing numeracy in embeddings. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

A Supplemental Material

A.1 Synthetic Numerical Data Generation

We briefly describe the numerical templates, providing the details missing from Table 2. In all cases, integers are sampled from $\{0, \ldots, 20K\}$, and split into disjoint train and development sets to assure generalization.

- *signed float combination* : Random signed combinations of up to 4 floats. Floats are sampled from the set of floats with two decimal places.
- *min/max/avg* : We sample 2-4 floats and apply a `min`, `max`, `avg`, operation by sampling a word from the set $\mathcal{O} = \{$*"longest"*, *"last"*, *"highest"*, *"largest"*, *"most"*, *"shortest"*, *"first"*, *"smallest"*, *"lowest"*, *"least"*, *"average"*$\}$.
- $\arg\max$, $\arg\min$: We sample word-float pairs, where words are sampled from $\mathcal{W}$: words in the NLTK Words Corpus[2] having at most 2 word-pieces, and floats are sampled as above.
- *date max/min* : Same as *min/max/avg* above, but for dates. Dates are sampled from $\mathcal{D}$: the set of dates until Sep 2019. The operator word is sampled from $\mathcal{DSUP} = \{$*"last"*, *"latest"*, *"most recent"*, *"youngest"*, *"first"*, *"earliest"*, *"oldest"*, *"least recent"*$\}$ and mapped to $\min$ or $\max$.
- *date difference* : This teaches our model to perform date arithmetic in days, months and years.
- *percentage* : We teach our model to perform $100 - x$ operations in the context of percentages. Given a number of arguments, we sample a percentage split using a flat Dirichlet distribution.

A.2 Synthetic Textual Data Generation

A.2.1 Sentence template extraction

To extract sentence templates, we abstract the text of math word problems from the corpus published by Hosseini et al. (2014). Going over examples, we split the problem text into sentences[3], and abstract the tokens of each sentence independently. Tokens are abstracted according to the framework into numbers (NUM), verb categories (VERB), entities (ENT), containers (CONT) and attributes (ATTR).

To have a better control over the generation process, we extend the framework of Hosseini et al. (2014) to support two container types - agent (AGT) and environment (ENV). Agents are objects which actively collect and drop entities, for example a person or an organization. Environments are passive containers, such as places or time periods. In addi-

tion, we introduce two-level containers to express inclusion relation between containers. For instance, if 3 submarines anchor near the city of Devonport, then they also anchor near the country of England.

The 12 most common extracted sentence templates, which were used for generating synthetic data, are provided in Table 8.

A.2.2 Template instantiation

Sentence templates are instantiated with a small vocabulary, that map categories into words. In this work, we construct two domain-specific small-world vocabularies, about history and the National Football League. The vocabularies are available in a json format in `https://github.com/ag1988/injecting_numeracy`.

A.2.3 Question templates

The 13 question templates for 7 different skills are provided in Table 9.

A.3 Data for Masked LM task

For creating the training data for the masked LM task (§ 5.1) we took the pages from English Wikipedia whose lowercased title containing a string in {*season, economy, demographics, conquest, war, battle, uprising, rebellion, insurgency, conflict, crisis, revolution, military history, mutiny, regiment, revolt, geography, raids, insurrection, invasion, feud, siege, campaign, expedition, succession, coup, university*}. This resulted in 156K full pages. In the remaining pages, paras with < 15 numbers were discarded. Pages were tokenized using DT (§ 3) and chunked into 512-token sequences. Following Devlin et al. (2019), each token was masked with probability 0.15 with no more than 65 masks per sample. This gave us 0.7M samples.

A.4 Experimental Setup

For all our experiments, we used an older version of Hugging Face's Transformers library (Wolf et al., 2019) and provide our training hyperparameters in Table 10.

A.5 GENBERT$_{+ND+TD}$ Error Analysis

Table 11 summarizes the main failure types of GEN-BERT$_{+ND+TD}$ on 100 random examples from the development set of DROP, excluding questions with a multi-span answer.

---

[2] `https://www.nltk.org/`

[3] Using the Spacy library `http://spacy.io/`

Template

```
CONT-1-AGT VERB-1-* NUM-1 ATTR-1 ENT-1 .
CONT-1-AGT VERB-1-POS NUM-1 ATTR-1 ENT-1 and CONT-2-AGT VERB-1-POS NUM-2 ATTR-1 ENT-1 .
CONT-1-AGT VERB-1-POS NUM-1 ATTR-1 ENT-1 and NUM-2 ATTR-2 ENT-2 .
CONT-1-AGT VERB-1-POS NUM-1 ATTR-1 ENT-1 , but VERB-2-NEG NUM-2 ATTR-2 ENT-2 .
CONT-1-AGT VERB-1-POS NUM-1 ATTR-1 ENT-1 in ATTR-2 CONT-2-ENV .
CONT-1-AGT VERB-1-NEG NUM-1 of the ATTR-1 ENT-1 .
CONT-1-AGT had NUM-1 ATTR-1 ENT-1 , CONT-2-AGT had NUM-2 ATTR-1 ENT-1 , and CONT-3-AGT had
NUM-3 ATTR-1 ENT-1 .
NUM-1 ATTR-1 ENT-1 , NUM-2 ATTR-2 ENT-2 , and NUM-3 ATTR-3 ENT-3 were VERB-1-POS in ATTR-4
CONT-1-ENV .
```
There were `NUM-1 ATTR-1 ENT-1` and `NUM-2 ATTR-2 ENT-2` in `ATTR-3 CONT-1-ENV` .
There were `NUM-1 ATTR-1 ENT-1` in `ATTR-2 CONT-1-ENV` .
```
CONT-1-AGT VERB-1-NEGTRN NUM-1 ATTR-1 ENT-1 to CONT-2-AGT .
CONT-1-AGT VERB-1-POSTRN NUM-1 ATTR-1 ENT-1 from CONT-2-AGT .
```

Table 8: Sentence templates for synthetic textual examples.

Reasoning	Templates
Selection	How many `ATTR-1 ENT-1` were in `CONT-1-ENV`?
	How many `ATTR-1 ENT-1` did `CONT-1-AGT VERB-POS`?
Intra-entity difference	How many more `ATTR-1 ENT-1` were in `CONT-1-ENV` than `ATTR-2 ENT-2` ?
	How many more `ATTR-1 ENT-1` did `CONT-1-AGT` have than `ATTR-2 ENT-2` ?
Intra-entity subset	How many `ENT-1` of `CONT-1` were `ATTR-1 ENT-1` ?
	How many `ENT-1` of `CONT-1` were not `ATTR-1 ENT-1` ?
Inter-entity comparison	Were there {more | less} `ATTR-1 ENT-1` in `CONT-1-ENV` or in `CONT-2-ENV` ?
	Who had {more | less} `ATTR-1 ENT-1`, `CONT-1-AGT` or `CONT-2-AGT` ?
Inter-entity superlative	Who had the {highest | lowest} number of `ATTR-1 ENT-1` in total ?
Intra-entity superlative	What was the {highest | lowest} number of `ATTR-1 ENT-1 VERB-POS` in `CONT-1-ENV` ?
	What is the {highest | lowest} number of `ATTR-1 ENT-1 CONT-1-AGT VERB-POS` ?
Inter-entity sum	How many `ATTR-1 ENT-1` were in `CONT-1-ENV` (, `CONT-*-ENV`) and `CONT-2-ENV` {in total | combined} ?
	How many `ATTR-1 ENT-1` did `CONT-1-ENV` (, `CONT-*-ENV`) and `CONT-2-ENV` have {in total | combined} ?

Table 9: Templates for questions about generated synthetic passages, testing for numerical reasoning. The template placeholders are filled-in with values from the world state obtained after generating the synthetic passage.

	pre-training			finetuning	
	lr	bsz	epochs	lr	bsz
GENBERT	-	-	-	3e-5	16
GENBERT+ND	6e-5	800	60	3e-5	16
GENBERT+ND-LM	4e-5	440	60	3e-5	16
GENBERT+ND-LM-DT	4e-5	440	60	-	-
GENBERT+ND-LM-RS	4e-5	440	60	3e-5	16
GENBERT+TD	1e-5	240	5	3e-5	14
GENBERT+ND+TD	1e-5	240	5	3e-5	14

Table 10: Hyperparameters used for pre-training GENBERT and finetuning it on DROP. lr=leaning rate, bsz=train batch size. Common params: seed=42, optimizer=Bert-Adam, linear-lr-warm-up=0.1, num epochs for finetuning=30, weight-decay=0.01, max-grad-norm=1.0.

Error category	Example
Counting	**q**: *How many people were the heads of the business?*
Sorting	**q**: *Which nationality was the fourth largest?*
Complex calculation	**q**: *How many percent of people were either Black Hispanic, of Sub-Saharan African origin, or of West Indian or Afro-Caribbean American origin?*
Complex semantics	**q**: *By how many points did the Pistons lose their closest game?*
Not numerical	**q**: *Who defeated the Kievan Rus at the Battle of the Alta River?*
Longer span	**q**: *Where there more people in the peninsula pre-war or at the time of the first census?* **a**: pre-war **p**: pre-war population
Shorter span	**q**: *Was the life expectancy in 2015 higher for males or females?* **a**: females **p**: female
Imprecise number prediction	**q**: *How many more estimated Chinese Americans lived in California compared to Massachusetts?* **a**: 1130100 **p**: 110100

Table 11: Error categories of GENBERT$_{+ND+TD}$ on the development set of DROP, based on a manual error analysis of 85 random examples. The upper part shows categories which are not not covered by our pre-training tasks or do not require numerical skills. The lower part shows categories of inaccurate model predictions. The letters **q**, **a** and **p** denote the question, gold answer and model prediction, respectively.