# NumGPT: Improving Numeracy Ability of Generative Pre-trained Models

**Zhihua Jin**[1] , **Xin Jiang**[2] , **Xingbo Wang**[1] , **Qun Liu**[2] , **Yong Wang**[3] , **Xiaozhe Ren**[2] and **Huamin Qu**[1]

[1]Hong Kong University of Science and Technology, Hong Kong
[2]Huawei Noah's Ark Lab
[3]Singapore Management University, Singapore

{zjinak,xwangeg,huamin}@ust.hk, {Jiang.Xin,qun.liu,renxiaozhe}@huawei.com, yongwang@smu.edu.sg

## Abstract

Existing generative pre-trained language models (e.g., GPT) focus on modeling the language structure and semantics of general texts. However, those models do not consider the numerical properties of numbers and cannot perform robustly on numerical reasoning tasks (e.g., math word problems and measurement estimation). In this paper, we propose NumGPT, a generative pre-trained model that explicitly models the numerical properties of numbers in texts. Specifically, it leverages a prototype-based numeral embedding to encode the *mantissa* of the number and an individual embedding to encode the *exponent* of the number. A numeral-aware loss function is designed to integrate numerals into the pre-training objective of NumGPT. We conduct extensive experiments on four different datasets to evaluate the numeracy ability of NumGPT. The experiment results show that NumGPT outperforms baseline models (e.g., GPT and GPT with DICE) on a range of numerical reasoning tasks such as measurement estimation, number comparison, math word problems, and magnitude classification. Ablation studies are also conducted to evaluate the impact of pre-training and model hyperparameters on the performance.

## 1 Introduction

Pre-trained models such as GPT [Radford *et al.*, 2018, 2019; Brown *et al.*, 2020] and BERT [Devlin *et al.*, 2019; Liu *et al.*, 2019; Lan *et al.*, 2019] have made remarkable achievements in natural language processing. Through fully utilizing a large-scale unlabeled corpus, they have successfully gained state-of-the-art results in various kinds of natural language understanding tasks, such as GLUE [Wang *et al.*, 2019b] and SuperGLUE [Wang *et al.*, 2019a].

Despite their inspiring performance in natural language understanding, these pre-trained language models still cannot do consistently well with tasks involving numbers [Thawani *et al.*, 2021]. For example, as shown in Fig. 1, the confidence for GPT answering the questions related to the weight of an egg tends to oscillate with the changes of answers. It exposes that GPT does not fully learn the continuous property of the
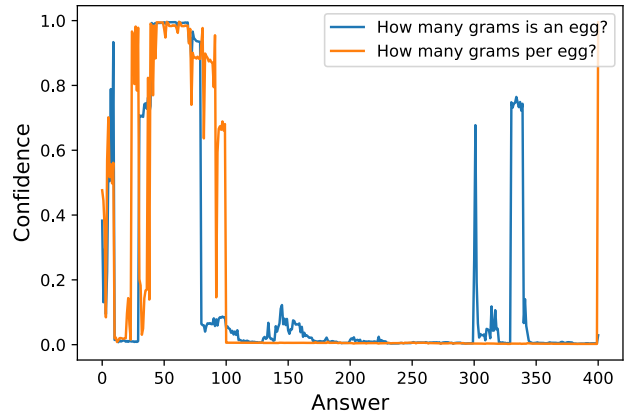


Figure 1: The confidence for GPT answering the questions related to the weight of an egg. The fluctuated curve reflects that GPT does not capture the continuous property of numbers.

numbers regarding certain context. Some researchers [Wallace *et al.*, 2019; Thawani *et al.*, 2021] regard it as *numeracy ability*. Their research has also confirmed that the large pre-trained model still cannot handle numerical information very well. The lacking of numeracy ability hinders those models from performing well on tasks requiring numeracy [Wallace *et al.*, 2019], which is prevalent in real world problems.

A line of work has proposed methods to improve the numeracy ability of the neural model. For example, Geva *et al.* [2020] improve the numeracy via generating more training data including numbers. Zhang *et al.* [2020] transform the number in the text into a scientific notation form. However, it does not guarantee that the models can learn numerical knowledge from the perspective of magnitude and mathematical notation. Other studies attempt to design a numeral embedding to improve the numeracy ability. They demonstrate that their designed embeddings can perform well in probing tasks on numeracy [Jiang *et al.*, 2019; Sundararaman *et al.*, 2020]. However, none of them pre-trains their model with numeral embeddings on a large corpus in a self-supervised manner. Therefore, we want to seek suitable numeral embeddings and incorporate them in pre-training large-scale language models.

Inspired by previous work [Jiang *et al.*, 2019; Sundararaman *et al.*, 2020; Zhang *et al.*, 2020], we propose NumGPT, a general autoregressive language model that uses a prototype-

based embedding to encode the *mantissa* (or *significand*) of the number and separate embeddings to encode the *exponent* of the number. We believe the inductive bias introduced by the numerical embedding will allow the model to present the precision and magnitude of numbers separately and generalize better on the numeric related tasks. We also design a numeral-aware loss function which enables the model to generate numbers as well as regular text tokens. To evaluate the performance of NumGPT, we synthesize several classification tasks such as measurement estimation, number comparison, and simple arithmetic problems. We also conduct experiments on a real dataset which focuses on magnitude classification, called Numeracy-600K [Chen *et al.*, 2019] to evaluate the numeral predictive ability of our model. Experiments demonstrate that our methods outperform the baseline models including GPT and GPT with DICE [Sundararaman *et al.*, 2020]. We further evaluate the generated text of NumGPT on a subset of the math word problem dataset, showing that our methods can generate better numerals. Ablation studies on the effect of pre-training and model hyperparameters are conducted to test whether they have a large impact on the model performance.

Our contributions can be summarized as follows:

- We propose a novel NumGPT model, which integrates the specifically designed numeral representations and loss function into the GPT model.

- We evaluate the numeracy ability of models on a series of synthetic and real-world tasks, and NumGPT achieves superior performance among them.

## 2 Methods

We focus on improving the numeracy ability of GPT in the two stages of the training process, i.e., pre-training and fine-tuning. The workflow includes pre-processing the numerals in the corpus, pre-training NumGPT with a numeral-aware language modeling loss, and fine-tuning the model on the domain-specific dataset. In the following sections, we will introduce our methods including numeral parser, numeral embedding, the architecture of NumGPT, and numeral-aware loss function.

### 2.1 Numeral Parser

The first step is to pre-process the numerals in text. We design a simple numeral parser to transform the common numerals in the text to the required format. It will be marked and further encoded by numeral embeddings. In real-world data, there exist many kinds of numerals [Ravichander *et al.*, 2019] and we mainly focus on three typical types: number, number with commas, and percentage. For example, "2300", "2,300" should be parsed to 2300. "23%" should be parsed to 0.23. Besides, we adopt Byte-Pair Encoding (BPE) [Sennrich *et al.*, 2016] to tokenize the regular text. After pre-processing the data, the output includes a binary label with each element indicating whether the corresponding element is a (textual) token or a numeral.

### 2.2 Prototype-based Numeral Embedding

For GPT, the token IDs will be transformed to embeddings through an embedding matrix. However, in terms of numerals,

regular embeddings would not suffice. The reason is that numbers are usually continuous and follow ordered relations, for example, they can be sorted based on their magnitude. In regular GPT, numerals are segmented into one or more independent subword units, which makes it cumbersome to learn the correct relation between numerals. Therefore, we need to design specific representations for the numerals so as to model them in an approximately continuous space.

Recent research has proposed using a deterministic approach [Sundararaman *et al.*, 2020] or calculating the weighted average of prototype embedding to determine the embedding of numeral [Jiang *et al.*, 2019]. Another work transforms numbers into the form of scientific notation to improve the model capability of capturing scale information [Zhang *et al.*, 2020]. Inspired by these work, we design a hybrid approach to embed the numerals which can capture the scale as well as the precision in a continuous manner.

For a numeral $n$, we will first transform it into a scientific notation and determine its exponent $e(n) \in \mathbb{Z}$ and mantissa $f(n) \in (-10, 10)$, as shown in Equation 1.

$$n = 10^{e(n)} \times f(n). \tag{1}$$

For example, $-123$ can be transformed to $-1.23 \times 10^2$. Its exponent, denoted as $e(-123)$, is the 2, and its mantissa, denoted as $f(-123)$, is $-1.23$. Based on the mantissa and exponent of the number, we calculate the numeral embedding $\text{NE}(n) \in \mathbb{R}^d$, where $d$ is the dimension of numeral embedding. We encode the mantissa and exponent of the number separately, as shown in Equation 2 and Fig. 2(a).

$$\text{NE}(n) = [\text{NE}^e(e(n)), \text{NE}^f(f(n))], \tag{2}$$

where $\text{NE}^e(e(n)) \in \mathbb{R}^{d_e}$ is the exponent embedding, and $\text{NE}^f(f(n)) \in \mathbb{R}^{d_f}$ is the mantissa embedding. $d_f$ and $d_e$ are dimensions of mantissa and exponent embeddings respectively, and $d = d_e + d_f$. In the current implementation, we empirically set $d_e = d/4, d_f = 3d/4$. For the number $-123$, its embedding is $\text{NE}(-123) = [\text{NE}^e(2), \text{NE}^f(-1.23)]$. We use different strategies to embed mantissa and exponent in the numeral embedding.

Considering that the exponent is an integer, we associate a learnable embedding vector $\text{NE}^e(e(n)) \in \mathbb{R}^{d_e}$ to each integer in the typical range of common numbers, that is $\{-8, -7, .., 11, 12\}$. For the exponent larger than 12 or less than $-8$, we set it to $+\text{INF}$ and $-\text{INF}$ respectively, as the signs of overflow and underflow for the model.

Since the mantissa is a real number, we choose a different embedding function for it. Similar to previous work [Sundararaman *et al.*, 2020; Jiang *et al.*, 2019], we adopt the deterministic approach and the prototype-based method to define the mantissa embedding. We denote the prototypes as $\{q_i^f\}_{i=0}^{d_f-1}$ where $q_i^f \in [-10, 10]$. Each value of the mantissa embedding is calculated based on the distance between the mantissa and each individual prototype. The formula is shown in Equation 3.

$$\text{NE}_i^f(f(n)) = \exp\left(-\frac{\|f(n) - q_i^f\|^2}{\sigma^2}\right) \tag{3}$$
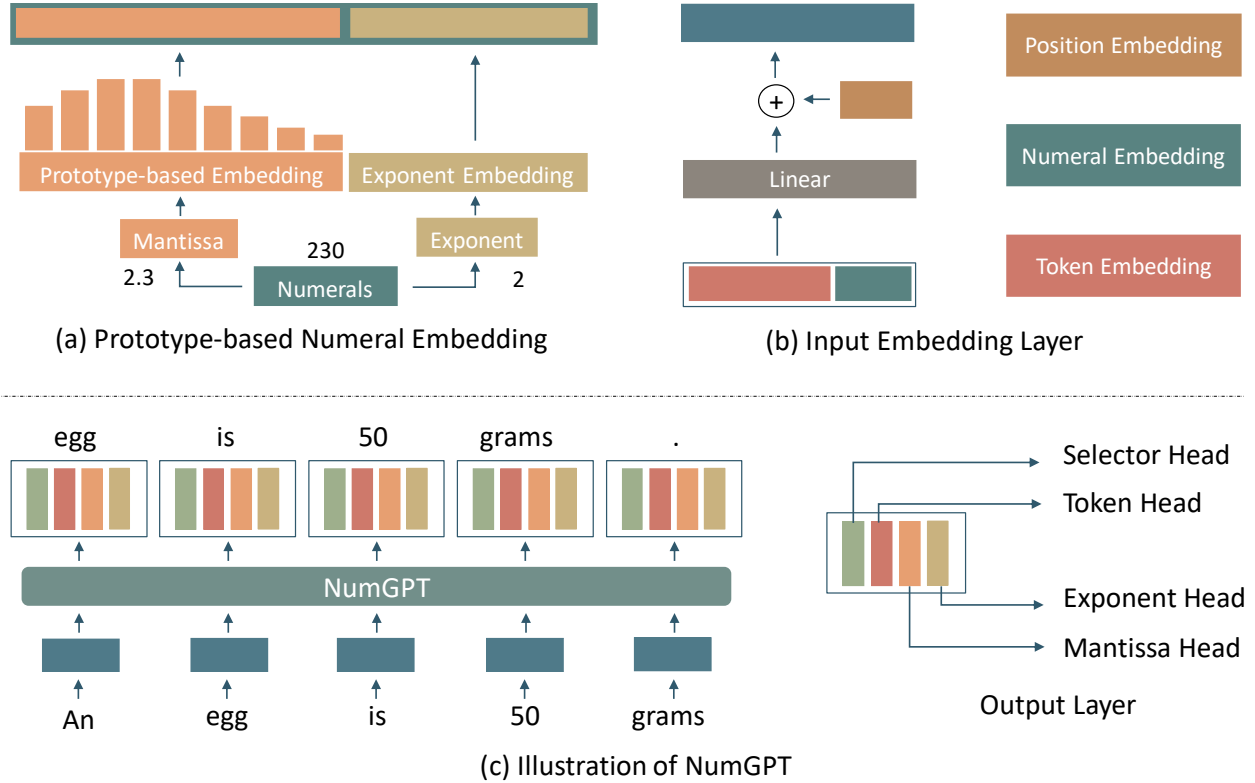
Figure 2: The model architecture of NumGPT: (a) We use prototype-based embedding to encode the mantissa of the numeral and use an individual embedding to encode the exponent of the numeral; (b) The numeral embedding and token embedding are fused in the input embedding layer; (c) NumGPT has four heads to control the output. If the Selector Head signals that it is a token, then it will use the Token Head to output a token. Otherwise, it will use Mantissa Head and Exponent Head to output a numeral.

where $\sigma$ is a hyperparameter controlling the smoothness of the mantissa representation. In this work, we simply use prototypes uniformly distributed in $[-10, 10]$:

$$q_i^f = \frac{10 - (-10)}{d^f - 1} \times i + (-10) \quad (4)$$

### 2.3 NumGPT

After deriving the numeral embedding, we can incorporate it into the pre-trained language models, or more specifically, GPT in this work. The architecture of GPT is based on the Transformer decoder. It can be formulated as three modules, i.e., the embedding layer, Transformer layers, and the output layer. The major difference between NumGPT, which is our proposed model, and GPT lies in the input embedding layer and the design of the output layer, as shown in Fig. 2(b) and Fig. 2(c) respectively.

**Input Embedding Layer.** In the design of the original Transformer [Vaswani *et al.*, 2017], it derives the input embedding through adding the token embedding and position embedding. Then it will be passed to the Transformer layer $h = \mathtt{Transformer}(\mathtt{E}(x_t)) \in \mathbb{R}^{d_h}$, where $d_h$ is the hidden size and $h$ is the hidden state. Consider the token embedding function as $\mathtt{TE}(x_t) \in \mathbb{R}^{d_h}$ and the position embedding function as $\mathtt{PE}(x_t) \in \mathbb{R}^{d_h}$, the output of input embedding layer

$\mathtt{E}(x_t) \in \mathbb{R}^{d_h}$ is shown in Equation 5:

$$\mathtt{E}(x_t) = \mathtt{TE}(x_t) + \mathtt{PE}(x_t), \quad (5)$$

where $t \in \{1, 2, .., T\}$ is the position in the sequence $x$ and $T$ is the length of sequence.

For NumGPT, we modify the input embedding layer to incorporate the numeral embedding for numbers. A paradigm is to concatenate the token embedding and the numeral embedding and then use a linear layer to transform it into the same size of hidden states. The formula is shown in Equation 6.

$$\mathtt{E}(x_t) = [\mathtt{TE}(x_t), \mathtt{NE}(x_t)]W_b + \mathtt{PE}(x_t), \quad (6)$$

where $W_b \in \mathbb{R}^{(d_h+d) \times d_h}$ denote parameters of the linear transformation. Since the token $x_t$ in the sequence can be textual or numeral exclusively, when the token is textual, its numeral part $\mathtt{NE}(x_t)$ becomes a learnable embedding shared among all the textual tokens. When the token is a numeral, the textual part $\mathtt{TE}(x_t)$ for the token becomes a learnable embedding shared across all the numerals.

**Output Layer.** In general, the output probability of the next token follows a mixture model:

$$\begin{aligned} p(x_t|&x_{1:t-1}) \\ =&p(z_t = 1|x_{1:t-1})p(x_t|x_{1:t-1}, z_t = 1) \\ &+ p(z_t = 0|x_{1:t-1})p(x_t|x_{1:t-1}, z_t = 0), \quad (7) \end{aligned}$$

where $z_t = 0$ indicates a textual token and $z_t = 1$ represents a numeral token. In the output layer, we decouple each part and model them separately. First, we use Selector Head to output the probability of token types. It can be calculated as follows:

$$p(z_t|x_{1:t-1}) = \texttt{softmax}(h_t W_z). \tag{8}$$

where $h_t \in \mathbb{R}^{d_h}$ is output hidden state and $W_z \in \mathbb{R}^{d_h \times 2}$ is the parameter matrix.

Then, if the Selector Head predicts the next token as textual, the Token Head will output the probability of the next tokens $\texttt{P}(h) \in \mathbb{R}^{T \times |V|}$, as shown in Equation 9.

$$p(x_t|x_{1:t-1}, z = 0) = \texttt{softmax}(h_t W), \tag{9}$$

where $W \in \mathbb{R}^{d_h \times |V|}$ is the parameter matrix, and $|V|$ is the size of vocabulary.

For a numeral token $x_t$, we decompose it into two parts $[x_t^e, x_t^f]$, which correspond to exponent and mantissa. They are regarded as independent variables and their marginal probabilities are modeled as:

$$p(x_t^e|x_{1:t-1}, z_t = 1) = \texttt{softmax}(h_t W_e) \tag{10}$$

$$p(x_t^f|x_{1:t-1}, z_t = 1) = \frac{1}{\sqrt{\pi}} \exp(-(x_t^f - h_t W_f)^2) \tag{11}$$

where $W_e \in \mathbb{R}^{d_h \times |V_e|}$ and $W_f \in \mathbb{R}^{d_h \times 1}$ are parameter matrices, and $|V_e|$ is the size of exponent vocabulary. $h_t W_f$ can be regarded as the mantissa of the predicted numeral.

## 2.4 Numeral-aware Loss Function

Finally, we obtain a numeral-aware language modeling loss function to pre-train the model:

$$
\begin{aligned}
\mathcal{L} = -\sum_{t=1}^{T} [&\log p(z_t|x_{1:t-1}) \\
&+ I(z_t = 0) \log p(x_t|x_{1:t-1}, z_t = 0) \\
&+ I(z_t = 1) \log p(x_t^e|x_{1:t-1}, z_t = 1) \\
&+ I(z_t = 1) \log p(x_t^f|x_{1:t-1}, z_t = 1)].
\end{aligned} \tag{12}
$$

## 3 Experiments

To evaluate the effectiveness of NumGPT in understanding numeracy, we conduct the following experiments. Inspired by previous evaluation methods [Hosseini *et al.*, 2014; Talmor *et al.*, 2020; Zhang *et al.*, 2020], we first synthesize three datasets as **Synthetic Tasks** to evaluate the numeracy ability of models. We further demonstrate the application of NumGPT in the magnitude classification task [Chen *et al.*, 2019]. Ablation studies are designed to investigate whether pre-training and the hyperparameters of NumGPT have a great impact on the performance of models.

## 3.1 Synthetic Tasks

We first compared NumGPT with baseline approaches, including GPT, GPT with DICE [Sundararaman *et al.*, 2020], on three synthetic tasks: Multiple Measurement Estimation (MME) task, General Number Comparison (GNC) task, and Math Word Problem Addition and Subtraction (MWPAS) task.

We create tens of templates for the tasks and sample random numbers to fill in the template. We frame those tasks as a binary classification problem, which judges whether the question-answer pair or the sentence is "correct" or not. We evaluate the model by calculating accuracy in the test dataset. We illustrate them one by one in the following paragraphs.

**Multiple Measurement Estimation.** Inspired by the measurement estimation task proposed by Zhang *et al.* [2020], we design the MME task to verify whether models can learn to estimate objects' numerical attributes. We construct a question-answer pair and let the model judge whether the answer matches the question. One question template is "How many grams are [INT] [OBJ]?" and we will sample a random integer to fill in "[INT]" and an object, such as "egg", to fill in "[OBJ]". We assume that the range of "correct" answer is $[\texttt{ANS\_MIN}, \texttt{ANS\_MAX}]$. We limit the range of "incorrect" answers in $[0.01 \times \texttt{ANS\_MIN}, \texttt{ANS\_MIN}) \cup (\texttt{ANS\_MAX}, 100 \times \texttt{ANS\_MAX}]$. For example, a question is "How many grams are 2 eggs?" and the answer range for this question is $[70, 140]$. If we input this question with the answer "35" to the model, the model should classify this question-answer pair as "incorrect". If the input answer is changed to "80", the model should classify this question-answer pair as "correct". For the dataset of this task, we have crafted 20 objects and 4 question templates. When generating one sample, we randomly sample a question template from candidate question templates, use the logarithmic sampler to sample the multiplier, and sample a candidate answer. For each object, we construct 500 "correct" samples and 500 "incorrect" samples. Then we split the generated samples into 16000 training samples and 4000 test samples. We guarantee that the combination of multipliers, object, and answer is unique for all the samples in the dataset.

**General Number Comparison.** Inspired by the number comparison task proposed in oLMpics [Talmor *et al.*, 2020], we augment it as general number comparison to evaluate whether the model can judge the quantitative comparison in the natural language context. A template for this problem is "A [NUMA] year old person is younger than a [NUMB] year old person", where "[NUMA]" and "[NUMB]" can be replaced with numbers in a range $[15, 104]$. The label is determined based on whether two numbers satisfy the semantics of the sentence. We have crafted 20 templates, which cover a large range of numbers and typical object numerical attributes (e.g., length and weight) comparison, and then according to each template, we generate 1000 positive samples and 1000 negative samples. In each sample, the number is randomly sampled using a logarithmic sampler to fill in the template. The dataset can be split into 32000 training samples and 8000 test samples.

**Math Word Problem Addition and Subtraction.** The MWPAS task assesses whether the model can handle the addition and subtraction task in the math word problem. Twenty templates are crafted from AI2 dataset [Hosseini *et al.*, 2014]. They cover the ability of addition and subtraction. For example, one template is "Joan found [NUMA] seashells on the beach. She gave Sam some of her seashells. She has [NUMB] seashells. How many seashells did she give to Sam?". Similarly, the "[NUMA]" and "[NUMB]" can be replaced by random integers. If the answer equals the subtraction from Number A to Number B, then the label is "true". Otherwise, it

| Model | MME | GNC | MWPAS |
|---|---|---|---|
| MAJ | 50.48±0.00 | 50.93±0.00 | 50.93±0.00 |
| *Train from scratch* | | | |
| GPT | 78.99±1.81 | 95.16±0.28 | 49.85±0.63 |
| GPT with DICE | 73.68±0.43 | 75.09±1.21 | 49.17±1.04 |
| NumGPT | 97.16±0.55 | 95.45±0.57 | **88.05**±0.99 |
| *Pre-train and finetune* | | | |
| GPT | 72.02±3.63 | 93.84±1.29 | 49.17±0.82 |
| NumGPT | **98.11**±0.38 | **95.82**±0.17 | 86.16±3.26 |

Table 1: Experiment results for synthetic tasks. The performance of NumGPT is better than baseline models in the three synthetic tasks ($p < 0.05$). The pre-training improves the performance of NumGPT on MME task ($p < 0.05$).

is "false". For each template, 1000 positive samples and 1000 negative samples are created. In each sample, numbers are randomly sampled using a logarithmic sampler to fill in the template. The splitting strategy is the same as the GNC task.
**Model Details.** In this study, we investigate the model performance on synthetic tasks. Baseline models include majority baseline (MAJ), GPT, and GPT with DICE. The MAJ baseline simply selects the most frequent label in the test dataset as the output. GPT is a Transformer decoder with 12 layers, 12 heads, and the hidden size of 768. NumGPT has the same architecture as GPT. The numeral embedding dimension of NumGPT is 64 and $\sigma$ is 0.5. For GPT with DICE, we replace the numeral embedding in NumGPT with DICE [Sundararaman *et al.*, 2020]. We train the models from scratch for 50 epochs on one Nvidia V100 GPU with batch size of 96. The optimizer we used is AdamW and the learning rate is $6.25 \times 10^{-5}$.
**Results.** The results of experiments on synthetic tasks are listed in Table 1. The mean and standard derivation of accuracy over 5 runs are reported. We also performed the Student's $t$-test to determine whether the average performance scores of two groups are significantly different. We assume that if the $p$-value is less than 0.05, the difference is statistically significant. NumGPT outperforms all the baseline methods in MME and MWPAS tasks ($p < 0.05$) and achieves the comparable results of GPT in GNC task. Those tasks focus on evaluating different aspects of the numeracy ability. For the MME task, we find that NumGPT can achieve a significant improvement compared to GPT and GPT with DICE ($p < 0.05$). It demonstrates that our numeral embedding captures the scale information more accurately, since it models the exponent individually. It further facilitates measurement estimation with multiplication, while GPT and GPT with DICE do not perform very well on it. For the GNC task, NumGPT has a slightly better performance than GPT. However, GPT with DICE cannot achieve a good performance on this task. One possible reason is that when integrating the DICE embedding into the model, the model cannot capture the tiny difference in the embedding if the numbers are very close. Therefore, it is hard for the model to distinguish similar numbers, which leads to a lower performance. For the MWPAS task, GPT and GPT with DICE have a similar performance as the MAJ baseline model. It demonstrates that they cannot model the addition and subtraction in the math word problems. NumGPT significantly

| Model | Micro-F1 ↑ | Macro-F1 ↑ |
|---|---|---|
| LR | 62.49 | 30.81 |
| CNN | 69.27 | 35.96 |
| GRU | 70.92 | 38.43 |
| BiGRU | 71.49 | 39.94 |
| CRNN | 69.50 | 36.15 |
| CNN-capsule | 63.11 | 29.41 |
| GRU-capsule | 70.73 | 33.57 |
| BiGRU-capsule | 71.49 | 34.18 |
| BiLSTM with DICE | 75.56 | 46.80 |
| *Train from scratch* | | |
| GPT | 79.45±0.45 | 53.86±0.84 |
| NumGPT | 78.97±0.24 | 53.30±1.12 |
| *Pre-train and finetune* | | |
| GPT | 79.42±0.15 | 53.79±0.68 |
| NumGPT | **81.32**±0.21 | **56.47**±0.77 |

Table 2: Experiment results on Numeracy-600K dataset. NumGPT outperforms other baseline models ($p < 0.05$).

outperforms other baseline models in this task ($p < 0.05$). It reflects that NumGPT has a more accurate arithmetic ability than other baseline models. A reason is that the numeral embedding of NumGPT eases the difficulty of modeling addition and subtraction.

## 3.2 Magnitude Classification

Magnitude classification on **Numeracy-600K** dataset [Chen *et al.*, 2019] is a task requiring models to predict the magnitude of the masked numeral in the news titles. We conduct the experiments on this dataset and compare the performance of our proposed model with that of the baselines.

**Model Details.** GPT is a Transformer decoder with 12 layers, 12 heads, and the hidden size of 768. NumGPT has the same hyperparameter configuration as GPT. We include the results of GPT and NumGPT. They are trained from scratch on one Nvidia V100 GPU with a batch size of 96 for 5 epochs. The optimizer used in this experiment is AdamW and the learning rate is set as $6.25 \times 10^{-5}$.

**Results.** Table 2 shows the experiment results on magnitude classification. Average values and standard deviations of Micro-F1 and Macro-F1 over 5 runs for GPT and NumGPT are reported respectively. Compared to the baseline results provided by Sundararaman *et al.* [2020], we can observe that our approach outperforms all the baseline models. Also, our model achieves a comparable performance with GPT. It indicates that NumGPT has a similar numeracy ability for magnitude prediction with GPT when training from scratch. A possible reason is that many numerals are masked in the input and NumGPT cannot show its representation power of numerical embeddings by only learning from the task dataset. While after pre-training the NumGPT on a large unlabeled corpus, it can achieve better performance than GPT on this task ($p < 0.05$). It demonstrates that through fully learning numeral relations in pre-training dataset, the performance of NumGPT can be improved on the downstream tasks.

## 3.3 Ablation Study

We want to identify whether the hyperparameters $\sigma$ used in the mantissa embedding will affect the model performance. Also, we would like to investigate whether the pre-training will have an impact on the model performance. We conduct both an ablation study regarding $\sigma$ on the MME task and another ablation study regarding pre-training on the synthetic tasks, and magnitude classification.
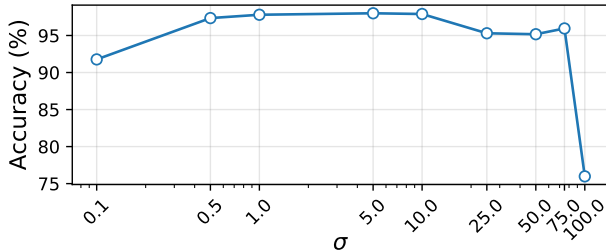


Figure 3: Experiment results for the ablation study on $\sigma$. The performance of NumGPT on the MME task will maintain in a small range except when $\sigma$ is very small, such as 0.1, or $\sigma$ is very large, such as 100.

**Ablation on $\sigma$.** We train NumGPT with $\sigma$ varying from 0.1 to 100. Other training settings are the same as the previous experiments on MME tasks. As shown in Fig. 3, we can observe that as the $\sigma$ increases, the performance is increasing and then decreasing after $\sigma$ is larger than 10. It seems that a very small $\sigma$ value, such as 0.1, or a very large $\sigma$ value, such as 100, will have a significant bad effect on the model performance. The reason is that when the value of $\sigma$ is very small, the numeral embedding will become sparse and then provide not enough detailed information to depict the numerals. When the value of $\sigma$ is very large, the numeral embedding will become very smooth and will not be able to distinguish between two mantissas. Therefore, we should choose a suitable $\sigma$ to help the model understand the mantissa of the numeral.

**Ablation on Pre-training.** For the pre-training, we pre-train GPT and NumGPT on the Wikipedia dataset, which contains 2,500M words, for one epoch on 8 Nvidia V100 GPUs with batch size 80. For the fine-tuning stage, we train the same epochs as training from scratch on the corresponding training dataset on one Nvidia V100 GPU with batch size 96. The experiment results are shown in Table 1 and Table 2. We repeatedly run the finetuning given the pre-trained weight in 5 runs and report the average and standard deviations of performance scores. The pre-trained NumGPT will improve the performance on MME task, and magnitude classification task ($p < 0.05$). After pre-training the models, in the magnitude classification task, NumGPT achieves more significant improvements than GPT ($p < 0.05$). It indicates that NumGPT can achieve better numeracy ability through pre-training on the large unlabeled corpus. While in GNC task and MW-PAS task, the performance improvements through pretraining on NumGPT are not significant. One possible reason is that the Wikipedia dataset does not have many samples requiring the arithmetic ability, such as numerical addition and subtraction required in the downstream tasks. We also find that pre-training cannot improve performance of GPT in synthetic tasks and magnitude classification. It reflects that it is hard for GPT to learn numeracy skills through pre-training in a large unlabeled corpus.

## 4 Related Work

We summarize related work into two categories: probing numeracy in NLP models and methods for improving numeracy ability of NLP models.

**Probing Numeracy in NLP Models.** Numeracy ability is mainly about reasoning with numbers in the text. The tasks for probing numeracy ability in NLP models can be further classified into two classes, approximate and exact, depending on the encoding of the numbers in text [Thawani *et al.*, 2021]. For example, "50" in the sentence "An egg weighs 50 grams." is an approximation of an egg weight, while the number "7" in the sentence "3 balls + 4 balls = 7 balls." is an exact answer. The probing tasks on numeracy for approximate numbers consist of numeration [Naik *et al.*, 2019; Wallace *et al.*, 2019], magnitude classification [Chen *et al.*, 2019], and measurement estimation [Zhang *et al.*, 2020]. The probing tasks on numeracy for exact numbers include number comparison [Talmor *et al.*, 2020] and math word problems [Ravichander *et al.*, 2019]. In this paper, similar to the previous tasks, we synthesize the measurement estimation task, the number comparison task, and math word problems. Besides, we adopt magnitude classification tasks [Chen *et al.*, 2019] to comprehensively evaluate the numeracy ability of models.

**Methods for Improving Numeracy Ability of NLP Models.** Researchers have explored some methods to improve the numeracy ability of NLP models. A line of work focuses on developing a domain-specific problem solver integrated with neural networks and symbolic functions for math word problems [Zhang *et al.*, 2019]. Their methods are mainly based on expression tree [Roy and Roth, 2015], sequence to sequence model [Wang *et al.*, 2017], reinforcement learning [Huang *et al.*, 2018; Wang *et al.*, 2018], and hybrid model [Amini *et al.*, 2019; Chiang and Chen, 2019; Griffith and Kalita, 2019]. Another line of work focuses on improving numeracy in word embeddings or pre-trained models [Jiang *et al.*, 2019; Sundararaman *et al.*, 2020; Berg-Kirkpatrick and Spokoyny, 2020], which can be generalized well across different tasks. Specifically, Jiang *et al.* [2019] proposed learning a weighted prototype numeral embedding and demonstrated that it can perform well on numeral prediction and sequence labeling tasks. However, the training process of the numeral embedding is not integrated into the training process of the models, which leads to suboptimal and time-consuming. Sundararaman *et al.* [2020] designed a deterministic corpus-independent numeral embedding with excellent performance on probing tasks for numeracy [Naik *et al.*, 2019; Wallace *et al.*, 2019]. However, when range of numbers becomes large, the embeddings for similar numbers will be very close and hard to distinguish. Moreover, in some tasks like math word problems, a subtle change in the number leads to totally different answers. While Berg-Kirkpatrick and Spokoyny [2020] conducted an empirical analysis of different loss functions in the task of contextualized numeral predictions in BERT, our work explores

how GPT can benefit from numeral-aware loss function, similar to their designed loss function. Particularly, we propose a deterministic numeral embedding and further integrate it in the pre-trained model GPT. Such a deterministic embedding considering the scientific notation of the numbers is more robust than the numeral embedding proposed by Sundararaman *et al.* [2020] for it is more scalable to produce embeddings for large numbers.

## 5 Conclusion

To improve the numeracy ability of GPT, we have proposed NumGPT, which incorporates the numeral embedding into the input embedding. For the numeral embedding, we used scientific notation to decompose the number into mantissa and exponent and embed them into separate parts. Moreover, we designed a numeral-aware loss to handle the generation of numeral. We have conducted the experiments to demonstrate the effectiveness of our method in the synthetic tasks and magnitude classification. We also conduct a series of ablation studies to test whether the hyperparameters and pre-training have a large impact on model performance. From the experiment results, the performance improvement encourages us that integrating numeral embedding into the GPT is a promising direction to improve the numeracy ability of language models.

In the future, we plan to extend our methods to a larger size model and conduct quantitative experiments on more numerical reasoning tasks. Also, the explanation of numerical capabilities of large-scale language models are still less explored. It will be interesting and valuable to further conduct research in future work.

## References

Aida Amini, Saadia Gabriel, Peter Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 2357–2367, 2019.

Taylor Berg-Kirkpatrick and Daniel Spokoyny. An empirical investigation of contextualized number prediction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 4754–4764, 2020.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

Chung-Chi Chen, Hen-Hsen Huang, Hiroya Takamura, and Hsin-Hsi Chen. Numeracy-600k: Learning numeracy for detecting exaggerated information in market comments. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6307–6313, 2019.

Ting-Rui Chiang and Yun-Nung Chen. Semantically-aligned equation generation for solving and reasoning math word problems. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 2656–2668, 2019.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, pages 4171–4186, 2019.

Mor Geva, Ankit Gupta, and Jonathan Berant. Injecting numerical reasoning skills into language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 946–958, 2020.

Kaden Griffith and Jugal Kalita. Solving arithmetic word problems automatically using transformer and unambiguous representations. In *International Conference on Computational Science and Computational Intelligence*, pages 526–532. IEEE, 2019.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 523–533, 2014.

Danqing Huang, Jing Liu, Chin-Yew Lin, and Jian Yin. Neural math word problem solver with reinforcement learning. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 213–223, 2018.

Chengyue Jiang, Zhonglin Nian, Kaihao Guo, Shanbo Chu, Yinggong Zhao, Libin Shen, and Kewei Tu. Learning numeral embeddings. *arXiv preprint arXiv:2001.00003*, 2019.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2019.

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

Aakanksha Naik, Abhilasha Ravichander, Carolyn Rose, and Eduard Hovy. Exploring numeracy in word embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3374–3380, 2019.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the Conference on Empiri-*

*cal Methods in Natural Language Processing*, pages 1743–1752, 2015.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.

Dhanasekar Sundararaman, Shijing Si, Vivek Subramanian, Guoyin Wang, Devamanyu Hazarika, and Lawrence Carin. Methods for numeracy-preserving word embeddings. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 4742–4753, 2020.

Alon Talmor, Yanai Elazar, Yoav Goldberg, and Jonathan Berant. olmpics–on what language model pre-training captures. *Transactions of the Association for Computational Linguistics*, 8:743–758, 2020.

Avijit Thawani, Jay Pujara, Pedro A Szekely, and Filip Ilievski. Representing numbers in nlp: a survey and a vision. *arXiv preprint arXiv:2103.13136*, 2021.

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

Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. Do nlp models know numbers? Probing numeracy in embeddings. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 5306–5314, 2019.

Yan Wang, Xiaojiang Liu, and Shuming Shi. Deep neural solver for math word problems. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 845–854, 2017.

Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, pages 5545–5552, 2018.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, pages 3261–3275, 2019.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019.

Dongxiang Zhang, Lei Wang, Luming Zhang, Bing Tian Dai, and Heng Tao Shen. The gap of semantic parsing: A survey on automatic math word problem solvers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(9):2287–2305, 2019.

Xikun Zhang, Deepak Ramachandran, Ian Tenney, Yanai Elazar, and Dan Roth. Do language embeddings capture scales? In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 4889–4896, 2020.