symmetry

MDPI

Article

Floating-Point Embedding: Enhancing the Mathematical Comprehension of Large Language Models

Xiaoxiao Jin , Chenyang Mao, Dengfeng Yue and Tuo Leng *

Department of Computer Engineering and Science, Shanghai University, Shanghai 200444, China; jxxxx6@shu.edu.cn (X.J.); shadymao@shu.edu.cn (C.M.); ydf@shu.edu.cn (D.Y.)

* Correspondence: tleng@shu.edu.cn

Abstract: The processing and comprehension of numerical information in natural language represent pivotal focal points of scholarly inquiry. Across diverse applications spanning text analysis to information retrieval, the adept management and understanding of the numerical content within natural language are indispensable in achieving task success. Specialized encoding and embedding techniques tailored to numerical data offer an avenue toward improved performance in tasks involving masked prediction and numerical reasoning, inherently characterized by numerical values. Consequently, treating numbers in text merely as words is inadequate; their numerical semantics must be underscored. Recent years have witnessed the emergence of a range of specific encoding methodologies designed explicitly for numerical content, demonstrating promising outcomes. We observe similarities between the Transformer architecture and CPU architecture, with symmetry playing a crucial role. In light of this observation and drawing inspiration from computer system theory, we introduce a floating-point representation and devise a corresponding embedding module. The numerical representations correspond one-to-one with their semantic vector values, rendering both symmetric regarding intermediate transformation methods. Our proposed methodology facilitates the more comprehensive encoding and embedding of numerical information within a predefined precision range, thereby ensuring a distinctive encoding representation for each numerical entity. Rigorous testing on multiple encoder-only models and datasets yielded results that stand out in terms of competitiveness. In comparison to the default embedding methods employed by models, our approach achieved an improvement of approximately 3.8% in Top-1 accuracy and a reduction in perplexity of approximately 0.43. These outcomes affirm the efficacy of our proposed method. Furthermore, the enrichment of numerical semantics through a more comprehensive embedding contributes to the augmentation of the model's capacity for semantic understanding.

Keywords: floating-point embedding; large language model; numerical semantics

Citation: Jin, X.; Mao, C.; Yue, D.; Leng, T. Floating-Point Embedding: Enhancing the Mathematical Comprehension of Large Language Models. *Symmetry* **2024**, *16*, 478. https://doi.org/10.3390/sym16040478

Academic Editor: Lorentz Jäntschi

Received: 10 March 2024
Revised: 8 April 2024
Accepted: 8 April 2024
Published: 15 April 2024

1. Introduction

The advent of the Transformer [1] has ushered in a revolutionary era in natural language processing research. This breakthrough has given rise to a series of pre-trained language models, including GPT [2] and BERT [3], which have demonstrated remarkable achievements in natural language processing. These models, trained on extensive unlabeled corpora, acquire knowledge of language rules, extract common-sense information embedded in text, and attain a generalized language representation, significantly elevating their performance across various downstream tasks [4–6].

The relationship between word embeddings and large language models (LLMs) is close. In the field of natural language processing (NLP), LLMs utilize word embeddings to construct their internal representations, thereby exhibiting remarkable performance in tasks such as text understanding and generation. This embedding technique enables LLMs to capture language semantics and contexts more effectively, offering more precise and flexible solutions for various NLP applications. Word embeddings provide a method of representing words within a vector space, enabling the capture of semantic relationships through both distance and direction. Their computation revolves around maximizing the likelihood of conditional probability distributions for each word in the lexicon. Recent advancements have introduced a novel class of word embeddings, drawing from concepts in information geometry, which have demonstrated the capacity to bolster the performance of various intrinsic and extrinsic NLP tasks [7]. However, the complexity of natural language, with its nuanced relationships, presents challenges. For instance, discerning contradictions within pairs of sentences may not be possible with conventional word embedding algorithms. Addressing this requires specialized neural networks tailored to learning embeddings attuned to contradictions [8]. Moreover, real-world linguistic environments, such as online forums and social media platforms, often introduce noisy text, which can undermine model performance. To mitigate this, the notion of bridge words has emerged—artificially introduced words intended to strengthen the similarity between standard words and their noisy variants [9]. Consequently, there is merit in exploring the fusion of traditional linguistic features with neural encodings. In tasks like predicting lexical complexity, leveraging Transformer models in conjunction with various traditional linguistic features has proven effective in enhancing the performance of deep learning systems [10]. Similarly, accommodating syntactic and morphological peculiarities is crucial, especially for languages like Welsh, a minority language, which necessitates adaptations to existing word embedding methods for optimal results [11]. Hence, the imperative lies in crafting specific word embedding methodologies tailored to the nuances of particular texts or tasks, recognizing the intricate interplay between linguistic structure and neural representations.

Notably, natural language text is replete with numerical elements. Despite the significant strides made in natural language understanding by LLMs, they encounter challenges in maintaining optimal performance when confronted with tasks involving numbers. This challenge primarily stems from the tokenization methods employed by these models, where numbers and words are treated uniformly. For instance, BERT utilizes a subword tokenization approach [12], potentially leading to the segmentation of a word into multiple parts. Take the number 87,600 as an example: it might be tokenized as 87 –600, disrupting the semantic coherence of the number itself. It is crucial to note that different tokenizers may yield distinct segmentations.

Certainly, language models default to interpreting numbers as entities akin to words. However, from the human perspective, numbers are perceived as integral, complete entities, and the human cognition of numbers involves a nuanced understanding. Numbers transcend the status of mere words within a text; they should be comprehended as mathematical entities and, furthermore, as symbols amenable to computation. The discussion here revolves around mathematical entities, focusing on their mathematical significance and computability. For instance, based on the current date and certain conditions, we can infer a new date that fits the context. However, for entities like phone numbers and license plates, inference is less applicable as they are more often considered identifiers rather than being viewed as mathematical entities. Therefore, it is imperative for language models to move beyond treating numbers as mere words. When numbers are acknowledged as mathematical entities, the model should discern and incorporate their inherent numerical semantics.

Recent studies have found that Chain of Thought (CoT) prompting [13–15] can significantly enhance the performance of LLMs, particularly in handling complex tasks involving mathematics or reasoning. However, the Chain of Thought method does not involve the special treatment of numbers; it simply improves the reasoning process. Fortunately, endeavors have been undertaken to augment the numerical understanding and fundamental arithmetic capabilities of such language models. The intrinsic numerical information embedded in numbers has been duly acknowledged, leading to the development of methodologies to encode and embed it as specific numerical representations fed into the model. One notably effective approach involves exponent embeddings through scientific notation, which folds numbers into evenly spaced bins on a logarithmic scale. Additionally, certain methodologies adopt binary-like encodings to represent the relative magnitudes of numbers within a specified real-number range.

Building upon a comprehensive review of prior research in this domain, we have assimilated insights from both theoretical understanding and experimental validation, resulting in a novel perspective. The Transformer architecture, upon closer examination, reveals parallels with conventional computer systems when viewed from a specific angle. Drawing an analogy between Transformer blocks and CPUs, the attention mechanism can be conceptualized as akin to addressing through both position and content. In this analogy, the feedforward network sub-layer and layer normalization layer mirror the functionalities of the arithmetic logic unit (ALU) in a CPU, conducting computations on the content addressed within the system.

In light of the identified parallels and symmetry between the Transformer architecture and CPU architecture, a natural question emerges: can we leverage traditional methods of handling numbers in computer systems to enhance the embedding process of Transformer architecture models? Within computer science, numerical representations within a certain precision range exhibit near-symmetry. Drawing inspiration from computer system theory [16–18], we applied the encoding method for floating-point numbers found in computer systems (specifically, the IEEE 754 standard) [19–21] to the Transformer architecture model. Subsequently, we devised a corresponding floating-point embedding module with the objective of enriching the model's understanding of numbers and effectively embedding their numerical features.

Our findings indicate that this floating-point embedding approach yields significant improvements, augmenting the model's comprehension of the numerical information within the text. We are the first to incorporate the floating-point representation method from computer systems into a deep-learning-based language model. We have designed a corresponding embedding module to process numerals in natural language text, thereby enhancing the model's understanding of the numerical information within the text. This work introduces a novel perspective and method for the exploration of numerical embeddings, opening avenues for future research in this domain. By extending the application of computer system principles, this approach contributes to the advancement of contemporary models.

In summary, our contributions can be outlined as follows.

- Identification of Transformer–CPU Connection: We establish a connection between the Transformer architecture and traditional computer systems by applying the floating-point representation method from computer systems to embed numerical information in the model. This involves the design of a corresponding embedding module to facilitate seamless integration.
- Validation of Floating-Point Embedding Efficacy: Through empirical testing on various encoder-only architectures, we demonstrate the effectiveness of the floating-point embedding approach. The results underscore its positive impact in enhancing the model's understanding of numbers and, consequently, improving task performance.

2. Related Work

In this section, our primary emphasis is on reviewing the diverse methods of numerical embedding, their application to encoders, and the extension of their use to decoder-only models. However, we refrain from delving into the discussion of numerical decoders.

Spithourakis et al. (2018) [22] and Wallace et al. (2019) [23] made attempts to amalgamate multiple numerical embeddings into a unified representation encapsulating the entire numerical value. Their approach involved training a language model to predict masked words and numbers, employing either recurrent neural networks (RNN) or convolutional neural networks (CNN) for pooling.

Sundararaman et al. (2020) [24] introduced a distinctive approach to numerical encoding termed Deterministic Independent Corpus Embeddings (DICE). This methodology aims to preserve the relative magnitudes of two numbers and their embeddings, utilizing cosine and Euclidean distances for encoding. Operating as a deterministic encoding, DICE can also serve as an auxiliary loss function.

Zhang et al. (2020) [25] adopted a unique approach by transforming all numbers in a dataset into scientific notation. They subsequently pre-trained a BERT model from scratch using this modified dataset, resulting in the creation of a new model named NumBERT. NumBERT retains the fundamental characteristics of the original BERT model but employs scientific notation for numerical representations.

Berg-Kirkpatrick et al. (2020) [26] conducted extensive research focused on predicting contextual numbers in text. The primary task involved predicting missing values and detecting incorrect numbers within sentences. Various numerical encoders were applied to the given numbers, and a set of output distributions was introduced to better align with the natural distribution of numbers in textual data.

Jiang et al. (2020) [27] introduced an innovative numerical embedding method that combines self-organizing maps or Gaussian mixture models, with the aim of improving the handling of out-of-vocabulary numbers. These numerical embeddings, trained through the skip-gram method, exhibit versatility and can be easily employed, enabling the training of multiple numerical encoders alongside word vectors. The experimental results illustrate the superior performance of such numerical embeddings in tasks related to numerical prediction and embedding.

Thawani et al. (2021) [28] provided a comprehensive summary of recent research on computational capabilities in natural language processing. The research underscores that the default subword tokenization using lookup embeddings to represent words is suboptimal for numbers. Additionally, Thawani et al. (2021) [29] further investigated and compared the impact of various numerical embedding methods on the reading and writing abilities of encoder-only models. The results indicate that, using BERT as an example, the introduction of numerical embeddings significantly enhances the encoder model's understanding of the numbers in the text.

Gorishniy et al. (2022) [30] conducted an investigation into numerical embedding methods specifically tailored to tabular data in table-based deep learning. Two approaches, one based on scalar value segmentation linear encoding and the other utilizing periodic activation, were explored. Both methods demonstrated significant performance improvements over traditional embeddings relying on conventional blocks like linear layers and ReLU activation. The findings underscore the benefits of embedding numerical features, emphasizing the significance of numerical feature embedding as a critical design aspect for various backbone networks.

Jin et al. (2022) [31] addressed a notable oversight in decoder-only models of the GPT series, which overlooked the numerical nature of digits. They introduced a novel numerical embedding method and devised a new model named NumGPT. Essentially, numbers were represented in exponential form, and a prototype-based numerical embedding was employed to encode the mantissa, with a single embedding to encode the exponent of the numbers. The results indicate that NumGPT, designed in this manner, surpasses conventional GPT models and GPT models combined with the DICE method in terms of computational capabilities.

3. Method

Our approach is grounded in the assumption proposed by Thawani et al. (2021) [29] that large language models (LLMs) benefit from numerical embeddings derived through specialized encoding based on the magnitudes of numbers. Additionally, we hypothesize the similarity and correlation between the architectures of LLMs and central processing units (CPUs) in computer systems. This similarity and correlation mainly manifest in the structure and functionality of the relevant neural network layers within the architectures of LLMs, as well as in the related functions and components of CPU architectures. The primary structure in LLMs is the Transformer block, each comprising a self-attention sub-layer and a feedforward neural network (FFN) sub-layer. The operation of the self-attention sub-layer is analogous to the CPU retrieving relevant data through instruction addressing. The FFN sub-layer consists of multilayer perceptrons and layer normalization layers, resembling the arithmetic logic unit (ALU) in a CPU, further processing the retrieved relevant data and producing the final output. Comparing the structure and functionality of the Transformer architecture to those of the CPU architecture, it can be observed that the Transformer model resembles a "novel" simplified computer system. A natural idea arises as to enabling Transformer models to encode and process numbers like computer systems. Consequently, we introduce floating-point representation from computer systems as a novel encoding method, followed by designing the corresponding embedding module. We emphasize the necessity of distinguishing between words and numbers in the text, applying specific encoding to generate embeddings, and then integrating them into the language model. In this section, we first introduce the floating-point representation and subsequently provide detailed insights into the corresponding embedding module.

we constructed our own model architecture. As illustrated in Figure 1, the backbone network remains an encoder model, as it is replaceable. The design of both the backbone network and the output draws mainly from the architecture of the BERT model [3], as our training task shares similarities with BERT model pre-training. Each input example contains only one number. This number is individually passed through our designed numerical embedding module to obtain its unique embedding representation. Other textual inputs still utilize the default embeddings of the encoder model.

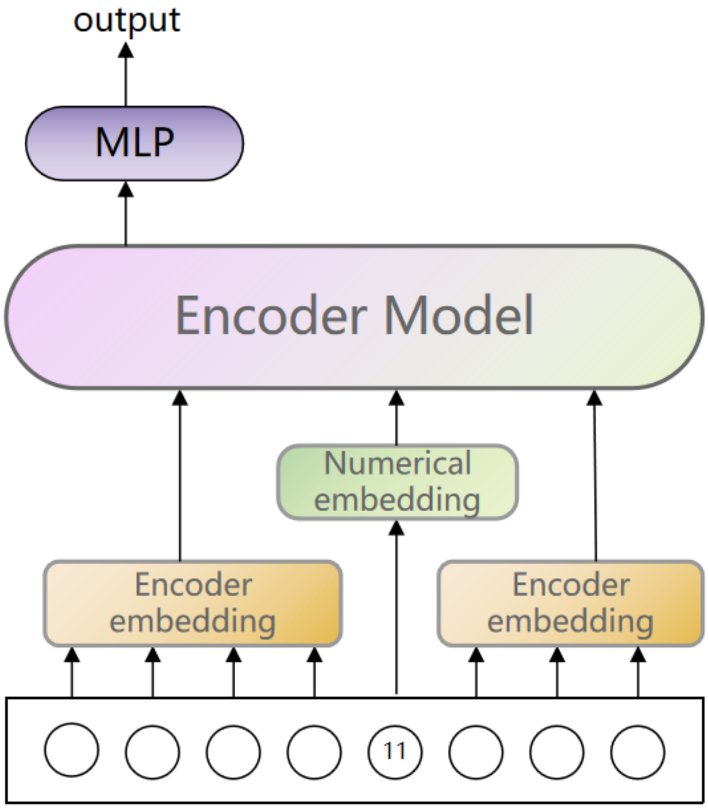

Figure 1. The architecture of our approach consists of an encoder-only model and a dedicated numerical embedding module designed for numerical values. Tokens other than numerical ones are embedded using the model's default embedding method. For the numerical token, a separate numerical embedding module is employed for embedding. The resulting embedded vectors are then input into the model to perform further calculations and generate an output. Each input example contains only a single numerical value. Finally, the model output is obtained through a multilayer perceptron (MLP) [32].

3.1. Floating-Point Representation (IEEE 754 Standard)

In computer systems, to represent very large and very small numbers without using an excessive number of zeros, floating-point numbers generally adopt the concept of scientific notation and can be expressed as

$$n = r^{E} \times M. \tag{1}$$

In (1), r is the base (typically 2), E is the exponent, and M is the mantissa.

In the IEEE 754 standard, a universal floating-point storage format is proposed based on the concept of scientific notation, outlining specifications for both single-precision (32-bit) and double-precision (64-bit) formats. Taking 32-bit floating-point numbers as an example, as shown in Figure 2, the 32 bits are subdivided into a sign bit (1 bit), exponent (8 bits), and mantissa (23 bits), with each bit expressed in binary form.

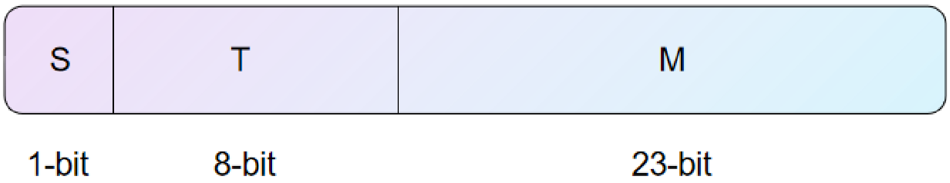

Figure 2. In the IEEE 754 standard, the representation of a 32-bit floating-point number is defined. Here, S represents the sign bit, T represents the exponent, and M represents the mantissa. Using −28.5 as an example, its corresponding 32-bit floating-point representation is 1 10000011 11001000000000000000000.

The sign bit denotes the sign of the number, where 0 represents positive and 1 represents negative. The exponent is presented in a biased form, signifying that the exponent equals the true value plus a bias. Typically, the bias is set to 2^{n-1}, and, in the IEEE 754 standard, it is specifically set to $2^{n-1} - 1$. For 32-bit floating-point numbers, the bias is established at 127. To ensure the unique representation of each number, normalization is applied to the numbers. Similar to scientific notation, the mantissa is in the form $1.M$, where $0 \leq M < 1$, and M represents the 23-bit mantissa in the IEEE 754 single-precision floating-point format.

Given that the first bit of the normalized mantissa is always 1, it does not need to be stored. Only the fractional part of M after the decimal point needs to be stored, and this M is represented using the two's complement notation. Therefore, under this standard, a floating-point number can be represented as

$$n = 1.M \times r^{T}. \tag{2}$$

When the mantissa M consists entirely of zeros and the true value of the exponent is -126 (i.e., the biased exponent is 00000001_B), the smallest absolute value that can be represented is $1.0_B \times 2^{-126}$. On the other hand, when the mantissa M consists entirely of ones, and the true value of the exponent is 127 (i.e., the biased exponent is 11111110_B), the largest absolute value that can be represented is $1.111\ldots_B \times 2^{127}$.

It is important to note that when both the exponent and the mantissa are all zeros, it represents the value 0. Conversely, when the exponent is all ones and the mantissa is all zeros, it represents either positive or negative infinity. When the exponent is all ones, but the mantissa is not all zeros, it signifies NaN (Not a Number).

In summary, the floating-point representation described can uniquely express a real number within a specific precision range as a binary sequence. This sequence not only captures the relative size relationships between numbers but also incorporates information about the absolute magnitude of the numbers. In essence, this encoding sequence comprehensively preserves numerical information and is decodable. This stands in contrast to previous approaches that utilized discretization encoding methods. Even when exponent methods were employed for encoding embeddings, they often mapped to an interval in a discretized manner, akin to dividing into buckets. As a result, while achieving effective numerical embeddings, these methods frequently sacrificed information about the absolute magnitude of the numbers. The adoption of the floating-point representation effectively addresses this limitation.

3.2. Floating-Point Embedding Module

Gorishniy et al.'s (2022) [30] research suggests that, with a designed numerical encoding, the introduction of a simple differentiable embedding module can further enhance the performance. The results demonstrate that, for tabular data, stacking two sets of linear layers and ReLU layers consecutively (referred to as the LRLR structure) has already yielded commendable results, albeit at a non-negligible cost [30]. We posit that the scale of the embedding module should be dependent on the parameter size of the chosen backbone network and the size of the dataset. Given that the parameter size of large language models (LLMs) typically exceeds 100 million, an embedding module with parameters exceeding 1 million is deemed reasonable.

Consequently, as shown in Figure 3, our floating-point embedding module consists of three consecutive sets of linear layers and ReLU layers (referred to as the LRLRLR structure). As we utilize 32-bit floating-point representation, the input dimension of the bottom-most linear layer is 32, while all remaining input and output dimensions align with the embedding vector dimension of the encoder model. Using BERT-base as an example, the input and output dimensions of all subsequent linear layers are set to 768. The bias of all linear layers is uniformly set to 0.

Our floating-point embedding module can be represented as follows:

$$\begin{gathered} x_1 = \mathrm{ReLU}(\mathrm{Linear}(x, W_1)), \\ x_2 = \mathrm{Linear}(\mathrm{ReLU}(\mathrm{Linear}(x_1, W_2)), W_3), \\ x_3 = x_1 + x_2, \\ embedding_{floating-point} = \mathrm{ReLU}(x_3). \end{gathered} \tag{3}$$

In (3), the input variable x has already undergone the standard conversion into a 32-bit floating-point representation. The ultimate output is an embedding vector, and the dimension of this vector is contingent on the embedding dimension employed by the model. This embedding vector serves as the numerical embedding representation input into the model.

Our belief is rooted in the notion that a floating-point representation can more comprehensively capture the numerical information of a number, ensuring that each representation for a number is unique. A simple representation can be adequately fitted with a straightforward function. However, as more information is included, a more complex function becomes necessary, rendering the embedding process correspondingly intricate. Consequently, the use of a relatively complex and deeper embedding module is justified in our approach.

Additionally, we introduce residual connections, facilitating the direct transfer of information across layers for more effective embeddings [33]. The incorporation of residual connections serves to expedite the convergence and enhance the training efficiency.

4. Experiments

To address our research question, we applied our floating-point embedding module for fine-tuning on four encoder-only pre-trained masked language models (BERT-base-uncased, BERT-large-uncased, Roberta-base, Roberta-large [34]). Simultaneously, we conducted comparisons with default tokenization methods and exponent embedding methods. Our focus was on investigating whether our floating-point embedding module could enhance the models' understanding of numbers across multiple architectures, providing a more comprehensive representation of numerical information and richer contextual semantics.

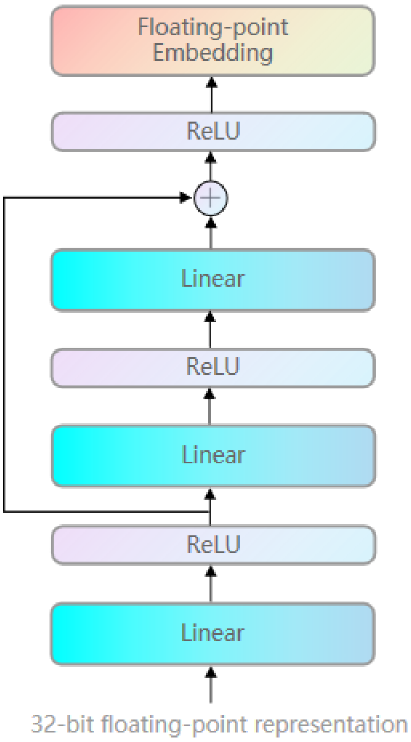

Figure 3. The network structure of the floating-point embedding module involves the input, which has already undergone standard conversion into a 32-bit floating-point representation, and the final output, which is an embedding vector.

The training and evaluation were performed on two datasets, Numerac y–600K [35] and Wiki-Concert [29]. Both datasets currently serve as baseline datasets for numerical understanding tasks. For both datasets, we conducted training on 100k samples, tested on 10k samples, and used an additional 10k samples as a validation set for hyperparameter tuning. The division of the datasets followed directly from the partitioning provided by the dataset constructors. To assess the model's understanding of numbers, we adhered to the pre-training paradigm during fine-tuning [3], ensuring task consistency. For each input sentence, we randomly masked 15% of non-numerical tokens and utilized negative log-likelihood loss to optimize the classifier. We measured the perplexity and Top-k accuracy, masking one (non-numerical) word at a time.

4.1. Datasets

Wiki-Concert consists of a carefully curated set of sentences where the numbers are manually annotated and sentences with multiple number annotations are filtered out. The dataset comprises over 900,000 sentences and annotated tuples.

Numeracy–600K is a large benchmark dataset that consists of two subsets. In financial documents, the in-depth analysis of both text and numerical information is required. This section primarily focuses on the financial market commentary dataset, with fields including "id" ,"unique_story_index", "offset", "length", and "magnitude". The second part is mainly collected from news data and constitutes a dataset of article headlines. Fields for this part include "id", "title", "publish_date", "offset", "length", and "magnitude". We utilized the first part for our experiments.

4.2. Baseline

Def. Default embeddings typically involve encoding numbers in encoder-only language models using common methods, such as subword tokenization, followed by lookup embeddings. This process treats numbers like words to derive embedding vectors for numerical representations.

Exp. Exponential embedding is an encoding method derived from scientific notation. We adopted the implementation approach outlined by Thawani et al. (2021) [29] . This method involves dividing the exponent part into n intervals on a logarithmic scale, which are then converted into corresponding lookup embeddings. This implementation can be extended to any number of intervals.

4.3. Implementation Details

We retrained the classification head and the corresponding numerical embedding module of an encoder-only language model while keeping the parameters of the language model itself frozen. The parameters of the language model were obtained from Hugging Face. The classification head of the encoder-only language model predicts the probability of masked words using the [CLS] token [3], which refers to a special token used in Transformer-based models in the model's output. The input and output dimensions of the network inside the classification head depend on the embedding vector dimension of the model and the number of words in the vocabulary. The encoding process of the floating-point representation was implemented using the 'struct' module. In the vocabulary of our encoder-only models, we introduced the token [NUM] to identify the unique numerical values in the substitute text. This inclusion is crucial as our methodology involves the individual encoding and embedding of numerical entities. Since each sample in the dataset contains only one numerical value, the model is designed to handle sentences with single numbers, and a special token [NUM] is introduced into the vocabulary to indicate the presence of a number in the sentence, enabling the model to accurately recognize and process numerical values. If a sentence contains multiple numbers, such a model design may encounter ambiguity in interpretation because it is unclear how to correctly handle multiple numbers. Therefore, to ensure the performance of the model and the accuracy of results, the design restricts each sentence to contain only one number. In future research, we will attempt to overcome this limitation. Thus, before applying our approach to instances with multiple number annotations, it is necessary to validate our method on tasks involving sentences with single numbers. As the model is designed to process sentences containing only a single number, it can only handle sentences with a single number. If a sentence contains multiple numbers, the model may encounter ambiguity in interpretation, as it is unclear how to properly handle multiple numbers. Therefore, to ensure the performance and accuracy of the model's results, our design restricts each sentence to contain only one number.

To assess the model's semantic understanding of numbers, we chose not to use the loss to predict the next sentence. Our training resources consisted of a single GeForce RTX 4090 GPU with 24 GB of VRAM. It takes less than 30 min on average to train a model for 15 epochs on 10k training samples. We set the batch size to 512, which is the maximum batch size that our GPU can accommodate for large models. We trained all models for 15 epochs on a training set containing 100k sentences. Smaller models converged well in 10 epochs, but, for large models, training for additional epochs further improved the convergence.

5. Results and Discussion

Table 1 displays the comparative results on the Numeracy–600K and Wiki-Concert datasets, showcasing a comparison between our method and the default embedding methods and exponential embedding methods. Table 2 presents the results of ablation experiments conducted for our floating-point embedding method. We conducted tests assessing the perplexity and prediction accuracy (Top-1, Top-5, Top-20, Top-100) on four models.

Table 1. Results on masked word prediction over two datasets, four models, and three embedding methods. PPL = perplexity. Def = default embeddings. Exp = exponent embeddings. Float = floating-point embeddings. The highlighted results represents the best results from each experiment group for each model. The downward arrow indicates that smaller results are preferable.

Model (Parameters)	Embedding	Wiki-Concert					Numeracy–600K				
		PPL ↓	Top-1	Top-5	Top-20	Top-100	PPL ↓	Top-1	Top-5	Top-20	Top-100
BERT-base-uncased (110 M)	def	2.993	66.977	84.693	91.415	95.829	3.651	66.004	79.604	86.807	92.534
	exp	2.954	67.417	84.978	91.438	96.105	3.527	66.763	80.301	87.244	92.815
	float	2.898	67.884	85.272	91.662	96.072	3.442	67.352	80.644	87.418	93.035
BERT-large-uncased (340 M)	def	2.62	70.184	87.027	92.85	96.357	3.28	68.323	81.563	88.166	93.504
	exp	2.605	70.311	87.038	92.857	96.337	3.079	69.6	82.552	88.962	93.939
	float	2.592	70.465	87.072	92.901	96.346	3.0	70.422	82.914	89.312	94.137
Roberta-base (125 M)	def	3.753	60.824	79.965	89.253	94.786	2.416	73.468	86.812	92.835	97.065
	exp	3.268	64.146	82.554	90.641	95.457	2.127	76.937	89.008	94.131	97.621
	float	3.326	64.06	82.283	90.496	95.337	2.098	77.276	89.245	94.236	97.659
Roberta-large (360 M)	def	2.828	67.142	85.742	92.399	96.391	1.73	83.057	92.596	96.081	98.362
	exp	2.62	69.544	86.651	93.053	96.723	1.691	83.802	93.046	96.277	98.456
	float	2.598	69.791	86.788	93.058	96.756	1.676	83.839	93.132	96.385	98.482

Our experiments consistently confirmed that the introduction of an additional embedding module for numbers significantly improved the model's semantic understanding of numerical information. In the majority of cases, our method demonstrated the best performance. On the Wiki-Concert dataset, our method did not surpass the exponential embedding method when applied to the Roberta-base model. However, it still achieved competitive results, enhancing the accuracy by approximately 3.2% and reducing the perplexity by about 0.42 compared to the default embedding method. We attribute this outcome to both methods drawing inspiration from the representation in scientific notation, leading to effective embeddings for numbers.

There were notable variations in the performance of the BERT and Roberta models on the two datasets, with BERT performing better on the Wiki-Concert dataset, while Roberta exhibited superior performance on the Numeracy–600K dataset. Moreover, the introduction of an additional numerical embedding module produced varied effects for both models. This phenomenon can be attributed to the distinct characteristics and distributions of the two datasets. Despite their similar sizes, if a dataset aligns more with the assumptions or is easier for a model to learn, the performance improvement on that dataset may be more pronounced. Roberta, as an improvement over BERT, undergoes more training steps and utilizes a larger pre-training dataset, generally making it more robust. It is worth noting that the Wiki-Concert dataset is primarily manually curated from Wikipedia, while Numeracy–600K consists of financial market commentary, leaning more towards raw data. This observation indirectly supports the idea that Roberta, with its enhanced ability to handle raw data, exhibits better generalization capabilities and adapts well to the characteristics of the data.

In conclusion, we observed an intriguing phenomenon: as the model size increased, i.e., when the number of parameters significantly rose, the incremental improvement brought about by the additional numerical embedding module became smaller. This suggests that with larger models, during the pre-training process, the model might have already learned some implicit numerical features and semantics, even when treating numbers as default embeddings similar to words. The era of diverse large models, exemplified by ChatGPT, is currently underway. Due to computational limitations, we did not conduct experiments on models such as LLAMA and ChatGLM. We speculate that the additional numerical embedding module might provide only marginal benefits for these types of models. Therefore, we posit that to further enhance the mathematical reasoning capabilities of large models, it may be necessary to introduce additional auxiliary modules specifically designed for reasoning tasks.

Table 2. Ablation experiment results on the floating-point embedding method using BERT-base-uncased as the backbone network for two datasets. Our approach excludes bias in the linear layer while incorporating residual connections. Additionally, we compare the results when introducing bias and removing residual connections from our method. PPL = perplexity. Float = floating-point embeddings. The highlighted results represents the best results from each experiment group for each model. The downward arrow indicates that smaller results are preferable.

Model	**Embedding**	**WikiConvert**			**Numeracy–600K**		
		PPL ↓	**Top-1**	**Top-5**	**PPL ↓**	**Top-1**	**Top-5**
BERT-base-uncased	float (w/bias)	3.125	66.003	83.996	3.572	66.592	79.941
	float (w/o residual)	2.931	67.621	84.968	3.47	67.242	80.489
	float (ours)	2.898	67.884	85.272	3.442	67.352	80.644

6. Conclusions

Our study delves deeper into the impact of incorporating a numerical embedding module into LLMs for the task of masked word prediction. The experimental results unequivocally demonstrate that our introduced floating-point representation and the corresponding floating-point embedding module significantly enhance the model's semantic understanding of numbers. We evaluate the performance primarily using the perplexity and Top-k accuracy.

We assert that our work provides additional evidence supporting the idea that thoughtfully designed numerical encoding methods and their corresponding embedding modules can elevate a model's semantic understanding capabilities. For relatively complex encoding methods, further research is warranted into optimizing the network architecture design of the embedding module. We aim to refine the network structure of the embedding module in future work. Our goal is to further validate that when the model processes sentences containing multiple numbers, applying our method leads to better understanding. This necessitates overcoming the current limitation of only being able to introduce a single special token [NUM] to mark numbers in the sentence. Additionally, our objective is to apply our method to network models with reasoning architectures, thereby enhancing the model's numerical reasoning abilities. Furthermore, we plan to validate our hypotheses by conducting experiments with our method on contemporary large-scale models.

Author Contributions: Conceptualization, X.J., C.M., D.Y. and T.L.; methodology, X.J., C.M., D.Y. and T.L.; code and experiments, X.J.; writing—original draft preparation, X.J.; writing—review and editing, X.J. and T.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the National Natural Science Foundation of China, grant 12071282.

Data Availability Statement: The datasets and models used in the experiments can be directly downloaded from the Hugging Face official website.

Acknowledgments: The authors are grateful to the reviewers for their valuable suggestions and comments.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*. [CrossRef]
2. Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I . Improving Language Understanding by Generative Pre-Training. 2018. https://api.semanticscholar.org/CorpusID:49313245 (accessed on 3 April 2024).

3. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the NAACL-HLT , Minneapolis, MN, USA, 2–7 June 2019; pp. 4171–4186.
4. Patil, R.; Gudivada, V. A Review of Current Trends, Techniques, and Challenges in Large Language Models (LLMs). *App. Sci.* **2024**, *14*, 2074. [CrossRef]
5. Cheng, J. Applications of Large Language Models in Pathology. *Bioengineering* **2024**, *11*, 342. [CrossRef]
6. Chow, J.C.; Wong, V.; Li, K. Generative Pre-Trained Transformer-Empowered Healthcare Conversations: Current Trends, Challenges, and Future Directions in Large Language Model-Enabled Medical Chatbots. *BioMedInformatics* **2024**, *4*, 837–852. [CrossRef]
7. Volpi, R.; Thakur, U.; Malagò, L. Changing the geometry of representations: α-embeddings for nlp tasks. *Entropy* **2021**, *23*, 287. [CrossRef] [PubMed]
8. Li, L.; Qin, B.; Liu, T. Contradiction detection with contradiction-specific word embedding. *Algorithms* **2017**, *10*, 59. [CrossRef]
9. Doval, Y.; Vilares, J.; Gómez-Rodríguez, C. Towards robust word embeddings for noisy texts. *App. Sci.* **2020**, *10*, 6893. [CrossRef]
10. Ortiz-Zambrano, J.A.; Espin-Riofrio, C.; Montejo-Ráez, A. Combining Transformer Embeddings with Linguistic Features for Complex Word Identification. *Electronics* **2022**, *12*, 120. [CrossRef]
11. Corcoran, P.; Palmer, G.; Arman, L.; Knight, D.; Spasić, I. Creating welsh language word embeddings. *App. Sci.* **2021**, *11*, 6896. [CrossRef]
12. Sennrich, R.; Haddow, B.; Birch, A. Neural Machine Translation of Rare Words with Subword Units. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Berlin, Germany, 7–12 August 2016; pp. 1715–1725.
13. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q.V.; Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 24824–24837.
14. Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.V.; Chi, E.H.; Narang, S.; Chowdhery, A.; Zhou, D. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In Proceedings of the Eleventh International Conference on Learning Representations, Online Event, 25–29 April 2022.
15. Feng, G.; Zhang, B.; Gu, Y.; Ye, H.; He, D.; Wang, L. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Adv. Neural Inf. Process. Syst.* **2024**, *36*. [CrossRef]
16. Chaudhuri, P.P. *Computer Organization and Design*; PHI Learning Pvt. Ltd.: Delhi, India, 2008; pp. 138–325.
17. Hennessy, J.L.; Patterson, D.A. *Computer Architecture: A Quantitative Approach*; Elsevier: Amsterdam, The Netherlands, 2011; pp. 2–61.
18. Bryant, R.E.; O'Hallaron, D.R. *Computer Systems: A Programmer's Perspective*; Prentice Hall: Hoboken, NJ, USA, 2011; pp. 37–179.
19. Kahan, W. IEEE standard 754 for binary floating-point arithmetic. *Lect. Notes Status IEEE* **1996**, *754*, 11.
20. Goldberg, D. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv. (CSUR)* **1991**, *23*, 5–48. [CrossRef]
21. Hough, D.G. The IEEE standard 754: One for the history books. *Computer* **2019**, *52*, 109–112. [CrossRef]
22. Spithourakis, G.; Riedel, S. Numeracy for Language Models: Evaluating and Improving their Ability to Predict Numbers. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Melbourne, Australia, 15–20 July 2018; pp. 2104–2115.
23. Wallace, E.; Wang, Y.; Li, S.; Singh, S.; Gardner, M. Do NLP Models Know Numbers? Probing Numeracy in Embeddings. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Hong Kong, China, 3–7 November 2019; pp. 5307–5315.
24. Sundararaman, D.; Si, S.; Subramanian, V.; Wang, G.; Hazarika, D.; Carin, L. Methods for numeracy-preserving word embeddings. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Online Event, 16–20 November 2020; pp. 4742–4753.
25. Zhang, X.; Ramachandran, D.; Tenney, I.; Elazar, Y.; Roth, D. Do Language Embeddings capture Scales? In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16–20 November 2020; pp. 4889–4896.
26. Berg-Kirkpatrick, T.; Spokoyny, D. An Empirical Investigation of Contextualized Number Prediction. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Online Event, 16–20 November 2020; pp. 4754–4764.
27. Jiang, C.; Nian, Z.; Guo, K.; Chu, S.; Zhao, Y.; Shen, L.; Tu, K. Learning numeral embedding. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16–20 November 2020; pp. 2586–2599.
28. Thawani, A.; Pujara, J.; Ilievski, F.; Szekely, P. Representing Numbers in NLP: a Survey and a Vision. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Online Event, 6–11 June 2021; pp. 644–656.
29. Thawani, A.; Pujara, J.; Ilievski, F. Numeracy enhances the literacy of language models. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, Punta Cana, Dominican Republic, 7–11 November 2021; pp. 6960–6967.
30. Gorishniy, Y.; Rubachev, I.; Babenko, A. On embeddings for numerical features in tabular deep learning. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 24991–25004.
31. Jin, Z.; Jiang, X.; Wang, X.; Liu, Q.; Wang, Y.; Ren, X.; Qu, H. NumGPT: Improving numeracy ability of generative pre-trained models. In Proceedings of the International Symposium on Large Language Models for Financial Services@ IJCAI, Macao, China, 20 August 2023.

32. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
33. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
34. Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv* **2019**, arXiv:1907.11692.
35. Chen, C.C.; Huang, H.H.; Takamura, H.; Chen, H.H. Numeracy-600K: Learning numeracy for detecting exaggerated information in market comments. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy, 28 July–2 August 2019; pp. 6307–6313.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.