Learning Numeracy: A Simple Yet Effective Number Embedding Approach Using Knowledge Graph

Hanyu Duan **Yi Yang** **Kar Yan Tam**
School of Business and Management, Hong Kong University of Science and Technology
hduanac@connect.ust.hk, {imyiyang,kytam}@ust.hk

Abstract

Numeracy plays a key role in natural language understanding. However, existing NLP approaches, either traditional word2vec approach or contextualized transformer-based language models, fail to learn numeracy. As the result, the performance of these models is limited when they are applied to number-intensive applications in clinical and financial domains. In this work, we propose a simple number embedding approach based on knowledge graph. We construct a knowledge graph consisting of number entities and magnitude relations. Knowledge graph embedding method is then applied to obtain number vectors. Our approach is easy to implement, and experiment results on various numeracy-related NLP tasks demonstrate the effectiveness and efficiency of our method.

1 Introduction

Numeracy is the ability to reason and to apply numerical concepts, and numbers play a key role in natural language understanding. For example, investors will probably react differently to the news "AAPL earnings increase by 2%" vs. "AAPL earnings increase by 20%". Similarly, in clinical setting, "heart rate is 140 beats per minute" vs. "heart rate is 60 beats per minute" will likely result in different decisions from physicians. In particular, healthcare providers often use the textual triage notes in emergency room to predict which patient to be discharged or admitted. Take a triage note for example, *"pt had unwitnessed GLF. was initially confused as per co-workers. GCS now 14. nauseated."* This note is labelled as Discharge as a GCS (i.e., Glasgow Coma Scale) of 14 indicates that the patient can response well (GCS ranges from 3 to 15, 3 being completely unresponsive and 15 being responsive). The number 14 plays a key role in the discharge decision. Using our proposed number embedding approach, the number is encoded into the same dimensional space as words while keeping the numeracy, so that we can use deep NLP model, say LSTM, to better represent the triage note. The model will explicitly learn that a GCS following a large number embedding may indicate Discharge while a GCS following a small number embedding may indicate Admitted. While numeracy is critical in such domains where numbers are prevalent, most existing NLP models are not designed explicitly to handle numbers. Numbers are either directly discarded in pre-processing, or treated as a `UNK` token (Thawani et al., 2021). Prior literature also shows that neither traditional word embeddings such as word2vec nor the contextualized transformer-based language model such as BERT can handle numbers and process numeracy tasks effectively (Naik et al., 2019; Wallace et al., 2019).

One straightforward way to encode numbers in NLP tasks is to map a number's value directly to its embedding (e.g., "twenty-four" embeds to [24]). Still, this strategy performs poorly while the NLP task involves a large amount of numbers with a wide range (Wallace et al., 2019). Therefore, encoding numbers into high-dimensional vector space may potentially overcome the difficulties and preserve numeric semantics. Along this line, Sundararaman et al. (2020) proposes a number embedding method DICE, where number vectors are obtained via mathematical operations. However, this method has a high computational cost due to the math operation which limits its use in encoding a large number of numbers.

In this work, we propose **NEKG** (**N**umber **E**mbeddings from **K**nowledge **G**raph), a simple yet effective number embedding method that produces numeracy-preserving embeddings via a knowledge graph structure. NEKG is independent of corpus and creates deterministic number embeddings. To explicitly preserve numeracy, we first construct a knowledge graph consisting of only number entities and magnitude relationships. We then apply TransE, a knowledge graph embedding method

Findings of the Association for Computational Linguistics: EMNLP 2021, pages 2597–2602
November 7–11, 2021.

Algorithm 1: Number Embeddings from Knowledge Graph

Input Number Embedding range $[min, max]$, embedding dimension dim, and the target number n.
Output The embedding vector $embd$ of the target number n.

//Constructing numerical knowledge graph.
loop
  $triple \leftarrow (h, "isLessThan", t)$ for each neighboring pair (h, t) within $[min, max]$
  append $triple \rightarrow Triples$
end loop
$model \leftarrow$ TransE ($k = dim$) // TransE is the graph embedding function obtained from AmpliGraph [1].
$model$.fit ($Triples$)
// Inferring embedding for a target number.
if $n \in$ OOV **:**
  $embd$ = Interpolation (n)
else :
  $embd$ = $model$.get_embeddings (n)

(Bordes et al., 2013), to embed the number entities into a vector space. In this way, numeracy-preserved embeddings can be obtained directly without using sophisticated mathematical operations. To obtain the embedding of an Out-of-Vocabulary (OOV) number, we propose an interpolation method that uses the weighted average of its two neighbors' embeddings based on cosine similarity. We experiment our method on several numeracy-related tasks, including evaluating embeddings on their ability to capture magnitude (Naik et al., 2019), and solving numeracy tasks (list maximum, decoding, and addition) (Wallace et al., 2019). We also apply our method in a downstream financial NLP task that predicts the magnitude of numbers in market comments (Chen et al., 2019). Experiments show that our approach is efficient, achieving comparable, and even better performance than existing numeracy-preserving methods.

2 Related Work

Numbers are ubiquitous and numeracy plays an important role in NLP applications and domains such as financial and clinical documents (Spithourakis et al., 2016; Rajkomar et al., 2018; Qin and Yang, 2019). However, most of existing work simply ignores the numbers in the pre-processing step (Kogan et al., 2009) and thus leads to suboptimal performance. See (Thawani et al., 2021) for an overview. Spithourakis and Riedel (2018) studies different strategies to model numerals, and Jiang et al. (2019) proposes a joint learning model for handling numbers in text. Still, a recent work (Naik et al., 2019) shows that common word embedding models cannot deal with numbers precisely. According to Wallace et al. (2019), most models fail to interpolate or extrapolate to OOV numerals. The main reason that causes such poor performance with number-intensive tasks is that the existing word embedding methods are not specifically designed to capture numerical relationships.

To handle number embeddings specifically, some new NLP models are proposed. One closely related work to ours is DICE (Sundararaman et al., 2020) which devises an independent-of-corpus and deterministic approach to assign embeddings for numbers. However, DICE derives numerical embedding based on engineered mathematical operations, which could be computationally costly for encoding a large number of numbers. Our work differs from DICE in that we infer numeracy-preserving embeddings automatically from a specially designed knowledge graph. Compared to DICE, our approach is simple and efficient yet achieves comparable or even better performance.

3 Methods

The high-level idea of our approach NEKG is to preserve numeracy and numeric semantics (e.g., magnitude, addition) via knowledge graph. Knowledge graph is a network of entities, their semantic types, properties, and relationships, built based on entity-relation triples (Popping, 2003). In our method, we make use of a simply structured knowledge graph consisting of only numbers and their magnitude relationships. We embed the knowledge graph in a vector space using a graph embedding method for obtaining embeddings of the number entities. Numbers that are not in the original knowledge graph, i.e., out-of-vocabulary (OOV) numbers, can be inferred by an interpolation method. See Algorithm 1 for the full descrip-

tion of our algorithm. The code is available at https://github.com/hduanac/NEKG.

3.1 Knowledge Graph Construction

The first step of our method is to construct a numerical knowledge graph. The reason we consider making use of a knowledge graph is based on the fact that knowledge graph can represent numbers and their magnitude relations properly and intuitively. We build the graph with a linear structure where entities are a sequence of ordered numbers. The minimal number and the maximal number in the graph are customized by the specific tasks. For example, suppose we are dealing with blood pressure numbers, we can set the minimum 0 and maximum 500 because blood pressures (in the units of mmHg) fall within this range. The entities of the graph are linked by a single relationship type called *"isLessThan"*, which ensures the transitive property of numbers can be captured. In other words, if a *"isLessThan"* b, and b *"isLessThan"* c, one can infer a *"isLessThan"* c from the knowledge graph. In Figure 1, we provide an illustration of the numerical knowledge graph with 11 numbers evenly sampled between 0 and 100. The range of the graph, i.e., the number of numeric entities, and minimal and maximal numbers, is determined by the downstream applications but it should have a linear structure and a single relationship type *"isLessThan"* for obtaining embeddings properly.

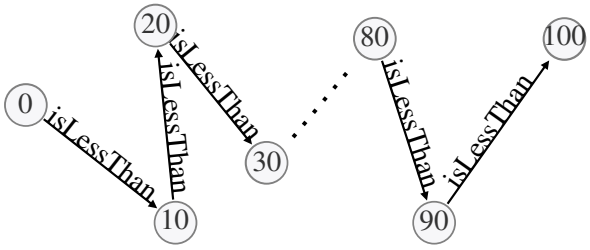

Figure 1: An illustration of numerical knowledge graph with 11 number entities $[0, 10, ..., 100]$.

3.2 Knowledge Graph Embedding

After constructing the knowledge graph, we embed the graph in a vector space using a standard graph embedding method TransE (Bordes et al., 2013). Other graph embedding methods can also be adopted, and we choose TransE for its simplicity and scalability. In TransE, if a fact *"subject, relationship, object"* holds, the embedding of the object entity should be close to the embedding of the subject entity plus the embedding that represents the relationship between them, i.e., $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$, and $\mathbf{h}$, $\mathbf{r}$, $\mathbf{t}$ stands for the embedding vectors of the triple (h, r, t), denoting a relationship r between entity h and t. In other words, our target is to get the embeddings of all the triples that satisfy $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$. In our context, $\mathbf{h}$ and $\mathbf{t}$ represent the embeddings of number entities, and $\mathbf{r}$ denotes the embedding of the relationship *"isLessThan"*. Therefore, we have $\mathbf{h}(0) + \mathbf{r}(\text{"}isLessThan\text{"}) \approx \mathbf{t}(10)$, $\mathbf{h}(10) + \mathbf{r}(\text{"}isLessThan\text{"}) \approx \mathbf{t}(20)$, $\mathbf{h}(20) + \mathbf{r}(\text{"}isLessThan\text{"}) \approx \mathbf{t}(30)$, etc. Then, intuitively, numbers with similar magnitude will have similar embeddings in the vector space. To validate it, we build a numerical knowledge graph with 200 integers ($[0, 199]$) and employ TransE to embed the graph in a 100-dimensional vector space. We visualize the embedding vectors of these 200 integers in Figure 2 using t-SNE (Van der Maaten and Hinton, 2008). The visualization shows that numbers with similar magnitude are close while numbers with different magnitude are further apart, in the learned embedding space.

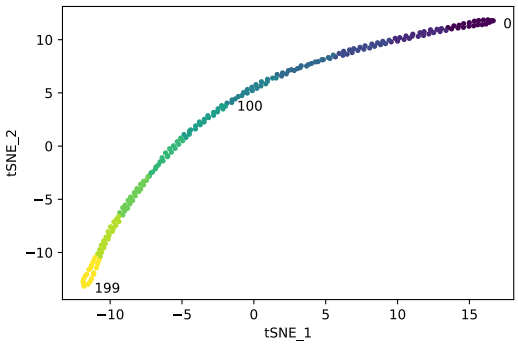

Figure 2: 2-D t-SNE visualization of 200 integers' embedding vectors. Darker dots indicate smaller numbers. Numbers with similar magnitude have similar number embeddings.

3.3 Interpolation Method for OOV Number

How can we obtain the embedding of a number that is *not* in the knowledge graph? We solve this OOV problem by an interpolation method based on the weighted average, where the embedding of a number is obtained by a weighted average of prototype embeddings (Jiang et al., 2019). In our work, we choose to use only two neighboring numbers as the prototype embeddings for simplicity. Suppose we have an OOV number n_0, we first locate two numbers n_1 and n_2 who are the two closest ones to n_0. Moreover, they satisfy $n_1 < n_0 < n_2$. We

then calculate their similarities as $s_1 = \frac{n_2 - n_0}{n_2 - n_1}$ and $s_2 = \frac{n_0 - n_1}{n_2 - n_1}$, where s_1 is the similarity between n_0 and n_1, and s_2 is the similarity between n_0 and n_2. Finally, the embedding of number n_0 is determined as $e_0 = s_1 \cdot e_1 + s_2 \cdot e_2$ where e_1 and e_2 are the embeddings of n_1 and n_2, respectively. According to this interpolation method, we can easily obtain the embedding of any number, integer or decimals.

4 Experiments

We evaluate NEKG on two tasks: **Task 1** evaluates NEKG on the ability to capture magnitude (Naik et al., 2019); **Task 2** examines the numeracy of NEKG on list maximum, decoding, and addition tasks (Wallace et al., 2019).

4.1 Task 1: Magnitude Contrastive Tests

In this task, we follow the analysis framework (Naik et al., 2019) to evaluate the ability of NEKG in capturing magnitudes by constructing contrastive tests (Zhu et al., 2018). The contrastive tests are based on triples (n, n^+, n^*). If the number embeddings can capture the magnitude, n will be closer to n^+ than n^* in the vector space, which means the embedding approach passes the test, and vice versa. As the previous work, there are three categories of the test (OVA, SC, and BC) that differ in the choice of n^*. The descriptions are as follows.

- **OVA (One-vs-All)**: The similarity between the embeddings of n and its nearest neighbor n^+ should be larger than that of the embeddings of n and n^*, where n^* stands for any other number in the dataset excluding n^+.
- **SC (Strict Contrastive)**: The similarity between the embeddings of n and its nearest neighbor n^+ should be larger than that of the embeddings of n and n^*, where n^* represents the second nearest neighbor of n.
- **BC (Broad Contrastive)**: The similarity between the embeddings of n and its nearest neighbor n^+ should be larger than that of the embeddings of n and n^*, where n^* is the furthest neighbor of n.

Training Details and Results. All the numbers are in the range of $[0, 9999]$ for both of the two tasks. We populate the triples of numbers randomly by following the rules of OVA, SC, and BC. Cosine similarity is used to measure the distance between two embedding vectors. We compare NEKG with a set of baseline methods, including **Random**: each number is represented by a random embedding; and **DICE** which is the state-of-the-art method that produces corpus-independent and deterministic number embeddings (Sundararaman et al., 2020). We implement embedding dimension $D = 100$ for NEKG. All the tests are performed on $10,000$ triples of numbers. The size of the knowledge graph is set as 100 (i.e., the number of entities). The performance is evaluated by accuracy, i.e., the percent of passed tests. The results in Table 1 show substantial improvement of our method over the baselines. Our model, which relies on simple knowledge graph embedding, achieves comparable performance with DICE. Besides, we compare the computational time cost of obtaining embeddings of numbers for NEKG and DICE in Figure 3. The result shows our method NEKG is 100-times faster (2-order of magnitude) than DICE method approximately. The high computational cost of DICE may be due to its sophisticated mathematical operations such as QR-decomposition and polar-to-Cartesian coordinate transformation. Therefore, NEKG will have more advantages while dealing with a large amount of data in NLP tasks.

Model	OVA	SC	BC
Random	0.73	49.29	50.63
DICE	99.46	99.68	99.78
NEKG (*Ours*)	**99.50**	**99.78**	**99.96**

Table 1: Performance (%) on magnitude contrastive tests.

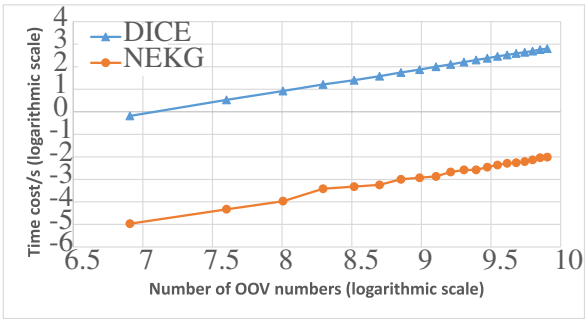

Figure 3: Time cost of DICE and NEKG in terms of obtaining embeddings of OOV numbers. For better comparison, we visualize it under a logarithmic scale.

[1] https://docs.ampligraph.org/en/1.3.0/generated/ampligraph.latent_features.TransE.html

	List Maximum (accuracy)			Decoding (RMSE)			Addition (RMSE)		
Integer range	[0, 99]	[0, 999]	[0, 9999]	[0, 99]	[0, 999]	[0, 9999]	[0, 99]	[0, 999]	[0, 9999]
Random vectors	0.92	0.69	0.68	24.79	290.32	2883.49	11.59	395.78	4136.60
DICE	0.97	**0.97**	0.96	0.42	0.81	2.98	0.63	3.72	28.50
NEKG (*Ours*)	**0.98**	0.96	**0.97**	**0.07**	**0.09**	**0.79**	**0.02**	**0.12**	**5.37**

Table 2: Experiment results on List Maximum, Decoding, and Addition.

4.2 Task 2: List Maximum, Decoding, and Addition

In task 2, we evaluate our method on several numeracy-tasks including List Maximum, Decoding, and Addition of numbers (Wallace et al., 2019). In particular, List Maximum is to predict the index of the maximal number in a list of the embeddings of five numbers. In Decoding, a number's embedding is given, we concern whether the model can regress the embedding of the number to its value. For Addition, given the embeddings of two numbers, the goal is to predict the sum of them.

Training Details and Results. Similar to Task 1, we compare our method NEKG with Random and DICE. For List Maximum, we feed the embeddings of five numbers through a Bi-LSTM network and use accuracy as the evaluation metric. For Decoding, we use a five-layer fully-connected neural network with ReLU activations to build a linear regression model trained by mean squared error (MSE) to regress the embedding of a number to its value. Root mean squared error (RMSE) is used for evaluation. For Addition, we concatenate the embeddings of two numbers and feed them through a five-layer fully-connected neural network with ReLU activations. Similar to Decoding, the model is trained by using MSE loss and evaluated by RMSE. The results in Table 2 show NEKG significantly outperforms other baselines in most cases, indicating a great understanding of numeracy.

5 Application of NEKG on Numeracy-600K

Numbers play an important role in financial documents. In this section, we evaluate NEKG in a real-world financial prediction dataset: Numeracy-600K (Chen et al., 2019). The Numeracy-600K task is a standard numeracy task studied in prior NLP research. The goal of the prediction task is to classify number magnitude in market comments. Specifically, given a sentence from market comments (e.g, AAPL price is up 2% to $142), the task aims to predict the magnitude of the target number, say 2%, using the surrounding texts and other numbers in the sentence.

Training Details and Results. The total number of magnitude classes in our experiment is 5. For example, the task defines the magnitude of numbers $0 \leq n < 1$ as 0, magnitude of numbers $1 \leq n < 10$ as 1, and so on. Following (Sundararaman et al., 2020), we implement a Bi-LSTM neural network to make predictions for target numbers that are masked by a certain random vector. The word embeddings are initialized with GloVe, and the numbers other than the target number in the sentence are initialized with NEKG or DICE. We include several baseline models where numbers are completely ignored, simulating a common practice of dealing with numbers in text. The prediction results are presented in Table 3, showing substantial improvement of our method in the F1 score, which validates the utility and effectiveness of our method in a downstream task. Comparing against baselines where numbers are ignored, the results highlight that incorporating numbers and number embeddings into NLP models can boost the prediction performance in financial applications.

Model	Micro-F1	Macro-F1
LR	41.55	35.32
CNN	43.27	47.25
BiLSTM	48.40	43.87
BiLSTM with DICE	59.55	**60.86**
BiLSTM with NEKG (*Ours*)	**60.66**	59.80

Table 3: Performance (%) on magnitude classification.

6 Conclusion

In this paper, we propose a simple yet effective number embedding method NEKG derived from knowledge graph. We evaluate our method on several numeracy-related tasks. The results show that NEKG can achieve better performance than the existing number embedding methods and run significantly faster. With the release of NEKG, we hope researchers and practitioners can utilize NEKG to handle numbers effectively in their work.

Impact Statement

Numeracy plays an important role in natural language understanding. Given the prevalence of numbers in real-world NLP applications in finance and healthcare domains, we hope the proposed number embedding method can be used as a plug-and-play tool in researchers and practitioners' NLP pipelines.

Acknowledgments

This work was supported by the FinTech Theme-based Research Scheme (No. T31-604/18-N) from Hong Kong Research Grants Council.

References

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Neural Information Processing Systems (NIPS)*, pages 1–9.

Chung-Chi Chen, Hen-Hsen Huang, Hiroya Takamura, and Hsin-Hsi Chen. 2019. 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.

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

Shimon Kogan, Dimitry Levin, Bryan R Routledge, Jacob S Sagi, and Noah A Smith. 2009. Predicting risk from financial reports with regression. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 272–280.

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

Roel Popping. 2003. Knowledge graphs and network text analysis. *Social Science Information*, 42(1):91–106.

Yu Qin and Yi Yang. 2019. What you say and how you say it matters: Predicting stock volatility using verbal and vocal cues. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 390–401.

Alvin Rajkomar, Eyal Oren, Kai Chen, Andrew M Dai, Nissan Hajaj, Michaela Hardt, Peter J Liu, Xiaobing Liu, Jake Marcus, Mimi Sun, et al. 2018. Scalable and accurate deep learning with electronic health records. *NPJ Digital Medicine*, 1(1):1–10.

Georgios P Spithourakis, Isabelle Augenstein, and Sebastian Riedel. 2016. Numerically grounded language models for semantic error correction. *arXiv preprint arXiv:1608.04147*.

Georgios P Spithourakis and Sebastian Riedel. 2018. Numeracy for language models: Evaluating and improving their ability to predict numbers. *arXiv preprint arXiv:1805.08154*.

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

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

Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. 2019. Do nlp models know numbers? probing numeracy in embeddings. *arXiv preprint arXiv:1909.07940*.

Xunjie Zhu, Tingfeng Li, and Gerard De Melo. 2018. Exploring semantic properties of sentence embeddings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 632–637.