LEARNING NUMERAL EMBEDDING

Anonymous authors
Paper under double-blind review

ABSTRACT

Word embedding is an essential building block for deep learning methods for natural language processing. Although word embedding has been extensively studied over the years, the problem of how to effectively embed numerals, a special subset of words, is still underexplored. Existing word embedding methods do not learn numeral embeddings well because there are an infinite number of numerals and their individual appearances in training corpora are highly scarce. In this paper, we propose two novel numeral embedding methods that can handle the out-of-vocabulary (OOV) problem for numerals. We first induce a finite set of prototype numerals using either a self-organizing map or a Gaussian mixture model. We then represent the embedding of a numeral as a weighted average of the prototype number embeddings. Numeral embeddings represented in this manner can be plugged into existing word embedding learning approaches such as skip-gram for training. We evaluated our methods and showed its effectiveness on four intrinsic and extrinsic tasks: word similarity, embedding numeracy, numeral prediction, and sequence labeling.

1 INTRODUCTION

Word embeddings, the distributed vector representations of words, have become the essential building block for deep learning approaches to natural language processing (NLP). The quality of pre-trained word embeddings has been shown to significantly impact the performance of neural approaches to a variety of NLP tasks. Over the past two decades, significant progress has been made in the development of word embedding techniques (Lund & Burgess, 1996; Bengio et al., 2003; Bullinaria & Levy, 2007; Mikolov et al., 2013b; Pennington et al., 2014). However, existing word embedding methods do not handle numerals adequately and cannot directly encode the numeracy and magnitude of a numeral (Naik et al., 2019). Most methods have a limited vocabulary size and therefore can only represent a small subset of the infinite number of numerals. Furthermore, most numerals have very scarce appearances in training corpora and therefore are more likely to be out-of-vocabulary (OOV) compared to non-numerical words. For example, numerals account for 6.15% of all unique tokens in English Wikipedia, but in GloVe (Pennington et al., 2014) which is partially trained on Wikipedia, only 3.79% of its vocabulary is numerals. Previous work (Spithourakis et al., 2016) also shows that the numeral OOV problem is even more severe when learning word embeddings from corpora with abundant numerals such as clinical reports. Even if a numeral is included in the vocabulary, its scarcity in the training corpus would negatively impact the learning accuracy of its embedding.

The inadequate handling of numerals in existing word embedding methods can be problematic in scenarios where numerals convey critical information. Take the following sentences for example,

“Jeff is 190, so he should wear size XXL” (190 is a reasonable height for size XXL)
“Jeff is 160, so he should wear size XXL” (160 is an unreasonable height for size XXL)
“Jeff is 10, so he should wear size XS” (10 is an age instead of a height)

If the numerals in the example are OOV or their embeddings are not accurately learned, then it becomes impossible to judge the categories of the numerals or the reasonableness of the sentences.

In this paper, we propose two novel methods that can produce reasonable embeddings for any numerals. The key idea is to represent the embedding of a numeral as a weighted average of a small set of prototype number embeddings. The prototype numerals are induced from the training corpus.
using either a self-organizing map (Kohonen, 1990) or a Gaussian mixture model. The weights are computed based on the differences between the target numeral and the prototype numerals, reflecting the inductive bias that numerals with similar quantities are likely to convey similar semantic information and thus should have similar embeddings. Numerical embeddings represented in this manner can then be plugged into a traditional word embedding method for training. We empirically evaluate our methods on four tasks: word similarity, embedding numeracy, numeral prediction, and sequence labeling. The results show that our methods can produce high-quality embeddings for both numerals and non-numerical words and improve the performance of downstream tasks.

2 RELATED WORK

Word Embedding Word embeddings are vector representations of words which carry semantic meanings implicitly and are trained without supervision. Most existing word embedding training methods can be divided into two classes. The first class of methods (Lund & Burgess, 1996; Rohde et al., 2006; Bullinaria & Levy, 2007; Lebret & Lebret, 2013) extract word co-occurrence statistics from the training corpus, compute a word-word matrix based on measures such as PPMI, and then apply dimension reduction techniques such as principle component analysis to produce a low-dimensional vector representation for each word. The second class of methods (Bengio et al., 2003; Collobert & Weston, 2008; Mikolov et al., 2013a,b) use a simple neural network to model the relation between a word and its context within a sliding window in the training corpus. GloVe (Pennington et al., 2014) has been proposed as a method that combines the advantages of both classes. All the above methods have a finite vocabulary size and use a ‘UNK’ symbol to represent OOV words. Recent work (Naik et al., 2019) shows that these popular methods do not handle numerals adequately. Wallace et al. (2019) shows that existing word embedding methods can encode numeracy implicitly for high-frequency numerals, but the embedding’s numeracy for OOV numerals is not investigated. Our goal is to design better numeral embedding methods that can be integrated into traditional word embedding methods and handle the OOV problem for numerals.

Numeracy in natural language Numerical understanding has been found important in textual entailment (Lev et al., 2004; De Marneffe et al., 2008; Roy et al., 2015) and information extraction (Intxaurrondo et al., 2015; Madaan et al., 2016), but existing systems often use manually defined task-specific features and logic rules to identify numerals, which is hard to generalize to other tasks. A lot of research has been done trying to solve math problems, using either manually designed features and rules (Roy et al., 2015; Mitra & Baral, 2016; Roy & Roth, 2016; Upadhyay et al., 2016) or sequence-to-sequence neural networks (Wang et al., 2017), but the quantity of numerals is not important in this task and hence existing methods often replace numerals by dummy symbols such as $n_1$ and $n_2$. Spithourakis & Riedel (2018) studied different strategies to better model numerals in language models. Chen et al. (2019) created Numeracy-600K dataset and studied the ability of neural network models to learn numeracy. Our work differs from previous work in that we aim to produce general-purpose numeral embeddings that can be employed in any neural NLP approach.

3 METHODS

Given a training corpus $C$, we first extract all the numerals using regular expressions and form a dataset $X$ containing all the numbers represented by these numerals. A number (e.g., 2000) may appear for multiple times in $X$ if its corresponding numerals (e.g., ‘2000’, ‘2,000’, etc.) appear for multiple times in $C$. We then induce a finite set $P$ of typical numerals (i.e., prototypes) from $X$ using a self-organizing map (Kohonen, 1990) or a Gaussian mixture model. We also define a function $sim(n_1, n_2)$ outputting the similarity between two arbitrary numbers $n_1$ and $n_2$. Now we represent the embedding of any target numeral $n$ as a weighted average of the prototype number embeddings with the weights computed by the similarity function:

$$e(n) = \sum_{p \in P}^{} \alpha \cdot sim(n, p) \cdot e(p), \quad \sum_{p \in P}^{} \alpha \cdot sim(n, p) = 1 \tag{1}$$

We use $e(\cdot)$ to denote the embedding of a number $\alpha$ is the normalization factor. This formulation satisfies the intuition that numerals with similar quantities are likely to convey similar semantic information and thus should have similar embeddings.
Our numeral embeddings can be integrated into traditional word embedding methods such as skip-gram for training. During training, we back-propagate the error gradient to update the prototype number embeddings. In this way, the prototype number embeddings (and hence all the numeral embeddings) are learned jointly with non-numerical word embeddings.

### 3.1 Squashing Numbers to Log-space

Inspired by psychological evidence that our brain compresses large quantities nonlinearly using a logarithmic scale on the mental number line (Nieder & Miller, 2003; Dehaene, 2011), we design the following squashing function to transform all the numbers in $X$ into the log-space before prototype induction. Alternatively, we can apply the function only in the similarity function.

$$f(x) = \begin{cases} \log(x) + 1, & \text{if } x > 1 \\ x, & \text{if } x \in [-1, 1] \\ -\log(-x) - 1, & \text{if } x < -1 \end{cases}$$  

### 3.2 Prototype Induction

We develop two methods for inducing a small prototype set $P$ from the number dataset $X$. Denote the number of prototypes by $m$.

**Self-Organizing Map** A self-organizing map (SOM) (Kohonen, 1990) is an artificial neural network that learns a discretized representation of the input space of the training samples in an unsupervised way. After training a SOM on the dataset $X$, each neuron in the SOM represents a prototype number. One advantage of using a SOM in comparison with traditional clustering methods is that it distributes prototypes more evenly on the number line and may assign prototypes to number ranges with few training samples, which we expect would lead to better generalizability.

**Gaussian Mixture Model** Inspired by psychological study of the mental number line (Dehaene et al., 2003) and previous work on language modeling (Spithourakis & Riedel, 2018), we train a Gaussian mixture model (GMM) to induce number prototypes. A GMM is defined as follows.

$$p(U = n) = \sum_{k=1}^{m} P(Z = k) P(U = n | Z = k) = \sum_{k=1}^{m} \pi_k N(n; \mu_k, \sigma_k^2)$$  

where $Z$ is a latent variable representing the mixture component for random variable $U$, and $N$ is the probability density function of a normal distribution, and $\pi_k, \mu_k, \sigma_k \in \mathbb{R}$ represent the mixing coefficient, mean and standard deviation of the $k$-th Gaussian component. We train a GMM on the number dataset $X$ using the expectation-maximization (EM) or hard-EM algorithm and regard the means of the learned Gaussian components as our prototypes $P = \{\mu_1, \cdots, \mu_m\}$. We use three GMM initialization methods described in Appendix A.

### 3.3 Similarity Function

For SOM-induced prototypes, we define the following similarity function:

$$\text{sim}(p, n) = |g(p) - g(n)|^{-\beta}, \quad \beta > 0, \quad p \in \mathbb{P}$$  

where function $g$ is equal to the squashing function $f$ defined in Eq.2 if we do not apply log transformation before prototype induction and is the identity function $I$ otherwise. $\beta$ is a hyper-parameter set to 1.0 by default.

For GMM-induced prototypes, we can naturally use the posterior probability of the component assignment to define the similarity function.

$$\text{sim}(p_k, n) \propto P(Z = k | U = n) = \frac{\pi_k N(n; \mu_k, \sigma_k^2)}{\sum_{k=1}^{m} \pi_k N(n; \mu_k, \sigma_k^2)}, \quad p_k \in \mathbb{P}$$  

### 3.4 Embedding Training

We now describe how to integrate our numeral embeddings into traditional word embedding methods for training. We choose skip-gram with negative sampling (Mikolov et al., 2013a,b) as the word
Figure 1: The computational graph when the center word is ‘is’ and the context words are ‘he’ and the numeral ‘190’. We look up the embedding vectors of non-numerical words directly from the embedding matrices and use the weighted average of prototype embeddings as the numeral embedding. Negative sampling is not shown in the figure.

embedding method here, but many other word embedding methods such as CBOW (Mikolov et al., 2013a), HAL (Lund & Burgess, 1996) and GloVe (Pennington et al., 2014) can be used as well.

Skip-gram is a word embedding method based on the idea of context word prediction. The training corpus \( C \) is regarded as a sequence of words \((x_1, \ldots, x_T)\). For token \( x_t \), we define the preceding and following \( c \) tokens as the context of \( x_t \). Skip-gram aims to maximize 

\[
\log p(x_{t+j}|x_t) \approx \log \sigma(v_{x_t+j}^o v_{x_t}^i) + \sum_{i=1}^{k} \mathbb{E}_{x_i \sim P_n(x)}[\log \sigma(-v_{x_t+i}^o v_{x_t}^i)]
\]

where \( \sigma \) denotes the sigmoid function, and \( P_n(x) \) is the negative word sampling distribution used to draw \( k \) negative samples.

We modify skip-gram by computing numeral embeddings differently from non-numerical word embeddings. We associate each prototype number with an input embedding and an output embedding. The input and output embeddings of all the prototypes constitute matrices \( M_I \in \mathbb{R}^{D \times |P|} \) and \( M_O \in \mathbb{R}^{D \times |P|} \) respectively. For any numeral, we can compute its input and output embeddings by taking a weighted average of the prototype input and output embeddings respectively based on Eq.1 and use them in exactly the same way as the embeddings of non-numerical words to compute the learning objective (Eq.5). When drawing negative samples, we first set the ratio of numerals and non-numerical words to their actual ratio in the training corpus, to guarantee a sufficient number of numeral negative samples. Then we sample numerals and non-numerical words separately from their respective distributions in the training corpus raised to the power of \( \frac{3}{4} \). During training, we optimize the objective function Eq.5 by back-propagating the gradient of error to update both the embedding matrices both the non-numerical word embedding matrices \( E_I, E_O \) and the prototype number embedding matrices \( M_I, M_O \). In this way, the embeddings of non-numerical words and numerals are learned jointly in the same space. We show an example in Figure 1.

4 Experiments and Results

We evaluate our methods on four intrinsic and extrinsic tasks: word similarity, embedding numeracy, numeral prediction, and sequence labeling. We report results of our methods based on SOM and GMM separately. We choose the hyper-parameters (e.g., the number of prototypes, GMM ini-
Non-numerical Word Vocabulary | Numerical Vocabulary
--- | ---
SOM, GMM, D-LSTM, Fixed | {In-vocab word}, UNK<sub>word</sub> | all numerals
NumAsTok | {In-vocab word}, UNK<sub>word</sub>, UNK<sub>num</sub>

Table 1: Vocabularies of different methods.

Table 2: Results on word similarity tasks trained on Wiki-1B. For reference, we also show the results of the official skip-gram and GloVe trained on larger corpora.

<table>
<thead>
<tr>
<th>Methods</th>
<th>WS353</th>
<th>MEN</th>
<th>SIM999</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOM</td>
<td>64.40</td>
<td>71.79</td>
<td>36.09</td>
</tr>
<tr>
<td>GMM</td>
<td>64.90</td>
<td>71.89</td>
<td>36.29</td>
</tr>
<tr>
<td>NumAsTok</td>
<td>65.30</td>
<td>71.83</td>
<td>35.85</td>
</tr>
<tr>
<td>D-LSTM</td>
<td>63.60</td>
<td>71.82</td>
<td>34.58</td>
</tr>
<tr>
<td>Fixed</td>
<td>64.35</td>
<td>72.17</td>
<td>36.27</td>
</tr>
<tr>
<td>SG GoogleNews-100B</td>
<td>70.00</td>
<td>74.10</td>
<td>44.20</td>
</tr>
<tr>
<td>GloVe Wiki-6B</td>
<td>52.20</td>
<td>73.70</td>
<td>37.10</td>
</tr>
</tbody>
</table>

4.1 Baselines

**NumAsTok** This baseline treats numerals and non-numerical words in the same way, which is very similar to the original skip-gram. The vocabulary includes both high-frequency words and high-frequency numerals. OOV non-numerical words are replaced with symbol UNK<sub>word</sub> and OOV numerals are replaced with symbol UNK<sub>num</sub>.

**D-LSTM** Character-level RNNs are often used to encode OOV words (Graves, 2013). Here we apply an LSTM (Hochreiter & Schmidhuber, 1997) to the digit sequence of a numeral and use the last hidden state of the LSTM as the embedding of the numeral. We use the embedding to compute the skip-gram objective function and propagate the gradients back to update the LSTM. The vocabulary of digits is: \{0-9, '+', '-', 'e'\}.

**Fixed** This baseline fixed embeddings for numerals with no training. We define the embedding a numeral with value \(n\) as \([f(n);1]/Z\) where \(f\) is the squashing function defined in Eq.\(2\), \(1 \in \mathbb{R}^{D-1}\) is an all-ones vector, and \(Z\) is a constant used to keep the vector norm close to those of non-numerical words and is set to \(2 \times D\) by default.

We compare the vocabularies of different methods in Table 1. Our methods, D-LSTM, and Fixed have finite non-numerical vocabularies but infinite numeral vocabularies. In contrast, the NumAsTok baseline has a finite numeral vocabulary and treats all the OOV numerals as UNK<sub>num</sub>.

4.2 Word Similarity for Non-numerical Words

To ensure that our methods can still generate high quality embeddings for non-numerical words, we evaluate our trained embeddings on classical intrinsic word similarity tasks, including WordSim-353, (Finkelstein et al. 2001), MEN (Bruni et al. 2014) and Simplex-999 (Hill et al. 2014). We train 300-dimensional word embeddings on the TB Wikipedia dump and set the context window size to 5, the number of negative samples to 5, and the vocabulary size to \(3 \times 10^5\). We use the evaluation tools\(^1\) provided by Jastrzebski (Jastrzebski et al. 2017). Note that while the training data contains numerals, the evaluation tasks do not involve numerals and are only designed to evaluate quality of non-numerical word embeddings. The results are shown in Table 2.

It can be seen that our methods can achieve scores comparable to those of the baselines. The performance of SG trained on 100B GoogleNews is much better than all the other methods probably because of its much larger training corpus. The results show that adding our numeral embedding methods into skip-gram does not harm the quality of non-numerical word embeddings. Additional results of our methods can be found in Appendix C.

\(^1\) https://github.com/kudkudak/word-embeddings-benchmarks
Table 3: Magnitude and numeration evaluation results for our methods and baselines. Accuracies of OVA, SC and BC are expressed as percentages. Lower AVGR indicates better performance. Numbers indicating top-2 performance are highlighted.

4.3 Magnitude and Numeration of Embeddings

Naik et al. (2019) propose a framework for evaluating the ability of numeral embeddings to capture magnitude and numeration. Given a target numeral, its embedding is evaluated against a set of numerals using the OVA (One-vs-All), SC (Strict Contrastive) and BC (Broad Contrastive) tests:

- **OVA**: The embedding vector distance between the target and its nearest neighbor on the number line should be smaller than that between the target and any other numeral in the set.
- **SC**: The embedding vector distance between the target and its nearest neighbor on the number line should be smaller than that between the target and its second nearest neighbors on the number line.
- **BC**: The embedding vector distance between the target and its nearest neighbor on the number line should be smaller than that between the target and its furthest neighbors on the number line.

We follow the settings described by Naik et al. (2019): for magnitude evaluation, we run the tests using a set of 2342 numerals sampled from Wikipedia-1B; and for numeration evaluation, we run the tests using 113 English words that represent numbers (e.g., ‘three’, ‘billion’) sampled from the same corpus and we measure the distance between the target numeral embedding and the word embeddings of these words. We report the accuracy of various embedding models on these three tests, along with the average rank (denoted as AVGR) of the target numeral’s nearest neighbor among all the candidates based on their vector distances to the target. We use the embeddings trained on Wikipedia-1B.

Table 3 shows the results. The Fixed baseline has the best performance in the magnitude evaluation, which is unsurprising because the numeral embedding vector explicitly contains the (squashed) magnitude. NumAsTok performs very well in the numeration evaluation, which is because the number-representing words used in the evaluation are mostly high-frequency words and their embeddings are adequately trained. Except for these two special cases, our methods can be seen to outperform the baselines with a large margin.

4.4 Numerical Prediction

To evaluate the quality of numeral embeddings, we design a new numeral prediction task: choosing the right numeral from a set of candidates given the context of the numeral in a sentence.

We randomly sample 2000 sentences containing numerals from a subset of Wikipedia that is not used in training, with 600 for validation and 1400 for testing. For each sentence, we use the five words preceding and following the target numeral as its context. An example is shown below, where the ten bold words are the context and 2.31 is the target numeral.

In Hollywood, the average household size was [2.31] and the average family size was 3.00.

We use all the 1400 numerals in the test set as the candidates from which one has to select the right numeral for each test sentence. Given the learned word and numeral embeddings, we define two score functions to rank candidate numerals given the context. Following the skip-gram model, we first define the score of center numeral $n$ predicting context word $c_j$ as $s(c_j|n) = v^{c_j}_n^T v_n$ and the score of context word $c_j$ predicting the center numeral $n$ as $s(n|c_j) = v^{c_j}_n^T v_{c_j}$. Our first candidate-ranking score function $S_A$ is the sum of log probabilities of center numeral $n$ predicting each context.
The task is to assign a label to each word or numeral in the dataset indicating its information type. We show two examples below:

Word: O O O O O O O O O O

82 kg 177 cm what size shall I choose 82 177 what size?

4.5 Sequence Labeling on Customer Service Data

To verify the effectiveness of our methods in practice, we evaluate our methods with a sequence labeling task on a dataset of customer service chat log from an online apparel shopping website. This dataset contains a large number of numerals related to height, weight, foot length, etc., and therefore is a good testbed for evaluating numeral embeddings.

The task is to assign a label to each word or numeral in the dataset indicating its information type. We show two examples below:

Word: O O O O O O O O O O

82 kg 177 cm what size shall I choose 82 177 what size?
Table 5: The results of sequence labeling. We report the accuracy, precision, recall, F1 score for the original, augmented, and harder test sets with different training data sizes. Accuracy is in the token level and the other metrics are in the entity level.

<table>
<thead>
<tr>
<th>Method</th>
<th>Original</th>
<th>Augmented</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc</td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>GMM</td>
<td>97.12</td>
<td>91.19</td>
<td>90.46</td>
</tr>
<tr>
<td>SOM</td>
<td>97.04</td>
<td>90.74</td>
<td>90.45</td>
</tr>
<tr>
<td>D-LSTM</td>
<td>96.72</td>
<td>89.84</td>
<td>88.80</td>
</tr>
<tr>
<td>Fixed</td>
<td>95.75</td>
<td>86.19</td>
<td>87.42</td>
</tr>
<tr>
<td>NumAsTok</td>
<td>96.88</td>
<td>91.37</td>
<td>89.29</td>
</tr>
<tr>
<td>GMM</td>
<td>96.21</td>
<td>89.25</td>
<td>86.07</td>
</tr>
<tr>
<td>SOM</td>
<td>96.20</td>
<td>89.50</td>
<td>86.18</td>
</tr>
<tr>
<td>D-LSTM</td>
<td>95.55</td>
<td>86.83</td>
<td>83.88</td>
</tr>
<tr>
<td>Fixed</td>
<td>94.67</td>
<td>83.51</td>
<td>82.69</td>
</tr>
<tr>
<td>NumAsTok</td>
<td>95.58</td>
<td>89.18</td>
<td>83.55</td>
</tr>
<tr>
<td>GMM</td>
<td>93.43</td>
<td>82.36</td>
<td>75.01</td>
</tr>
<tr>
<td>SOM</td>
<td>93.48</td>
<td>82.13</td>
<td>75.11</td>
</tr>
<tr>
<td>D-LSTM</td>
<td>92.53</td>
<td>77.11</td>
<td>71.45</td>
</tr>
<tr>
<td>Fixed</td>
<td>91.90</td>
<td>75.39</td>
<td>71.41</td>
</tr>
<tr>
<td>NumAsTok</td>
<td>92.31</td>
<td>81.98</td>
<td>70.51</td>
</tr>
</tbody>
</table>

W, H, O are labels representing weight, height and ordinary word respectively. We show the statistics of the dataset in Appendix F. In order to better evaluate the generalizability, we create two additional test sets. The first one is created by ‘augmenting’ the original test set with new sentences containing slightly perturbed numerals. For example, we can create new sentences by replacing ‘177’ in the above example with ‘176’ and ‘178’. The second one contains ‘hard’ sentences from the original test set that do not have explicit cues for label prediction. For example, the first sentence above contains ‘kg’ and ‘cm’ that can greatly facilitate the prediction of W and H, but the second sentence above does not contain such cues and hence is a ‘hard’ sentence. More details about the two test sets can be found in Appendix E. Finally, we also test the low-resource settings in which only 30% or 10% of the training set is used.

We learn embeddings from the training set using our methods and the baselines and use a validation set to do model selection. We plug the learned embeddings into the Neural-CRF model (Yang & Zhang, 2018) to do sequence labeling without using part-of-speech and character-level features and embedding fine-tuning.

The results are shown in Table 5. Our methods consistently outperform all the baselines on the Accuracy, Recall, and F1 metrics in different configurations. NumAsTok trained with 100% training samples has the highest precision on the original and hard test sets probably because it learns high-quality embeddings for high-frequency numerals included in its vocabulary; but its recall is lower than that of our methods, most likely because of its numeral OOV problem. Comparing the results on the original and augmented test sets, we see that NumAsTok shows a more significant drop in performance than the other methods, which suggests that NumAsTok does not generalize well because of the numeral OOV problem. In the low-resource settings, the advantage of our methods over the baselines becomes even larger, indicating better generalizability and less annotation required for our methods to achieve a promising performance.

5 Conclusion

In this paper, we propose two novel numeral embedding methods that represents the embedding of a numeral as a weighted average of a set of prototype numeral embeddings. The methods can be integrated into traditional word embedding approaches such as skip-gram for training. We evaluate our methods on four intrinsic and extrinsic tasks, including word similarity, embedding numeracy, numeral prediction, and sequence labeling, and show that our methods can improve the performance of numeral-related tasks and has better generalizability. Our code and sample data can be found at path/to/code/.
REFERENCES


A GMM INITIALIZATION

Both EM and hard-EM are sensitive to initialization and we use the initialization methods described in (Blömer & Bujna, 2013). We first initialize the mean $\mu_k$ of the $k$-th Gaussian component using one of the following three strategies:

**Random initialization:** choose $\mu_k$ from $X$ randomly. This is suitable when $X$ contains a wide range of numbers, e.g., numbers collected from Wikipedia.

**SOM-based initialization:** initialize $\mu_k$ to $p_k \in \mathbb{P}$ produced by the SOM method.

**K-means initialization:** run randomly initialized k-means on $X$ and then use k-means centroids to initialize $\mu_k$.

We then assign the data samples to their closest means. The standard deviation of the data samples assigned to the $k$-th mean becomes $\sigma_k$.

B HYPER-PARAMETERS

We list all of the important hyper-parameters we tune for each model.

**General hyper-parameters:** embedding dimension, context window size, SGD learning rate, batch size, vocabulary size, etc.

**SOM hyper-parameters:** number of prototypes, stage of applying the log-squashing function (stage 1: before prototype induction; stage 2: only in the similarity function).

**GMM hyper-parameters:** number of prototypes, whether we apply the log-squashing function to the numerals, EM initialization (from SOM, random initialization, or k-means initialization), type of EM (hard-EM or soft-EM).

We show the values of the SOM and GMM hyper-parameters in Table 6 and the values of the general hyper-parameters of all the methods in Table 7. We find that the general hyper-parameters influence the performance of our methods and the baselines in the same way, so in most cases, these hyper-parameters are set to be identical for all the methods. For large training corpora (Wiki1B, Numeracy-600k), we use 2048 as the batch size for D-LSTM, because D-LSTM consumes much more GPU memory. We set the batch size of the other methods to 4096. For the sequence labeling tasks, because the data is relatively small and confined to a very specific domain (chat log from online apparel shops), we set a small vocabulary size of 500 for all the methods except NumAsTok and set the vocabulary size of NumAsTok to 550 to ensure that different methods have similar numbers of parameters for word embedding training. Consequently, our methods have $(500 + |\mathbb{P}|) \times D$ parameters for word embedding training and NumAsTok has $550 \times D$ parameters, where $\mathbb{P}$ is the prototype set, whose size is typically smaller than 50, and $D$ is the embedding dimension.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>SOM prototype number</th>
<th>GMM prototype number</th>
<th>SOM log transform stage</th>
<th>GMM log transform stage</th>
<th>SOM initialization</th>
<th>GMM initialization</th>
<th>EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word similarity (Wiki1B)</td>
<td>200</td>
<td>200</td>
<td>dataset</td>
<td>200</td>
<td>True</td>
<td>random</td>
<td>hard</td>
</tr>
<tr>
<td>Magnitude (Wiki1B)</td>
<td>200</td>
<td>200</td>
<td>dataset</td>
<td>300</td>
<td>True</td>
<td>random</td>
<td>soft</td>
</tr>
<tr>
<td>Numeration (Wiki1B)</td>
<td>300</td>
<td>300</td>
<td>similarity function</td>
<td>500</td>
<td>True</td>
<td>random</td>
<td>soft</td>
</tr>
<tr>
<td>Numerical Prediction (Wiki1B)</td>
<td>300</td>
<td>300</td>
<td>similarity function</td>
<td>300</td>
<td>False</td>
<td>random</td>
<td>hard</td>
</tr>
<tr>
<td>Sequence Labeling 100 %</td>
<td>15</td>
<td>15</td>
<td>dataset</td>
<td>200</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>Sequence Labeling 30 %</td>
<td>10</td>
<td>10</td>
<td>dataset</td>
<td>30</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>Sequence Labeling 10 %</td>
<td>25</td>
<td>25</td>
<td>similarity function</td>
<td>20</td>
<td>False</td>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

Table 6: Hyper-parameter values for GMM and SOM based methods for each experiment.
Table 7: Values of general hyper-parameters for each experiment.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Embedding dim</th>
<th>Context Window</th>
<th>Negative Samples</th>
<th>Epoch</th>
<th>Batch Size</th>
<th>Learning Rate</th>
<th>Vocabulary Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word similarity (Wiki1B)</td>
<td>300</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>4096, 2048</td>
<td>$5 \times 10^{-3}$</td>
<td>$3 \times 10^5$</td>
</tr>
<tr>
<td>Magnitude (-MAG) (Wiki1B)</td>
<td>300</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>4096, 2048</td>
<td>$5 \times 10^{-3}$</td>
<td>$3 \times 10^5$</td>
</tr>
<tr>
<td>Numeration (-NUM) (Wiki1B)</td>
<td>300</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>4096, 2048</td>
<td>$5 \times 10^{-3}$</td>
<td>$3 \times 10^5$</td>
</tr>
<tr>
<td>Numeral Prediction (Wiki1B)</td>
<td>300</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>4096, 2048</td>
<td>$5 \times 10^{-3}$</td>
<td>$3 \times 10^5$</td>
</tr>
<tr>
<td>Numeral Prediction (Numeracy-600k)</td>
<td>300</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>4096, 2048</td>
<td>$5 \times 10^{-3}$</td>
<td>$1 \times 10^5$</td>
</tr>
<tr>
<td>Sequence Labeling 100% 30% 10%</td>
<td>50</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>50</td>
<td>$5 \times 10^{-2}$</td>
<td>500, 550</td>
</tr>
</tbody>
</table>

Figure 2: Histograms of numerals and learned prototypes that range from $0$ to $10^{13}$. The horizontal axis represents the numeral quantity and the vertical axis represents the number of occurrences, ‘500’ means the number of prototypes, ‘soft’ means soft-EM.

C More results on Wikipedia-1B

We show the histograms of numerals in the Wikipedia-1B dataset and the prototypes learned by SOM and GMM in Fig.2. It can be seen that the prototypes induced by our methods have a similar distribution compared to the original numerals.

We also show some examples of prototypes and their nearest non-numerical words in Table 8. We use the embedding trained by the SOM model with 200 prototypes on Wikipedia-1B, and use log transformation in the similarity function.

<table>
<thead>
<tr>
<th>Prototype Value</th>
<th>Most Related Non-numerical Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>8186446.58</td>
<td>million, billion, total, budget, funding, estimated, dollars</td>
</tr>
<tr>
<td>10372.49</td>
<td>thousand, approximately, thousands, millions, roughly, hundreds</td>
</tr>
<tr>
<td>2000.06</td>
<td>nearly, millennium, decade, internet, twentieth, worldwide, latest</td>
</tr>
<tr>
<td>1598.79</td>
<td>johann, renaissance, giovanni, dutch, baroque, vii, shakespeare</td>
</tr>
<tr>
<td>10.00</td>
<td>ten, six, eleven, pm, seconds, eight</td>
</tr>
</tbody>
</table>

Table 8: Examples of prototypes and their nearest non-numerical words.

In addition, we select several typical numerals and non-numerical words and project their embeddings to 2D using t-SNE (Figure 3). We use embeddings learned on Wikipedia-1B corpus using the SOM and GMM methods. The examples and the figures show that our model does capture some semantic relations between numeral quantities and normal words.

<table>
<thead>
<tr>
<th>Method</th>
<th>SOM</th>
<th>GMM</th>
<th>NumAsTok</th>
<th>D-LSTM</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed (sent/s)</td>
<td>13590.93</td>
<td>12691.18</td>
<td><strong>22907.97</strong></td>
<td>8421.66</td>
<td>13055.08</td>
</tr>
</tbody>
</table>

Table 9: Training speed for each methods.

We show the training speed of each embedding method on the Wikipedia-1B dataset in Table 9. The batch size is set to 2048 for all the methods. Our methods are slower than NumAsTok but are faster than D-LSTM.
(a) t-SNE plot for embedding trained by the SOM-based method with 200 prototypes.

(b) t-SNE plot for embedding trained by the GMM-based method with 300 prototypes, random initialization and soft-EM training.

Figure 3: 2D t-SNE results for the SOM-based and GMM-based methods.
Table 10: Statistics of low-resource customer-service dataset.

### D Numerical Prediction Evaluation Metrics

We denote the target numeral by $n_i$, the numeral with the highest ranking score by $\hat{n}_i$, and the rank of the target numeral by $r_i$. The error $e_i$ and percentage error $pe_i$ can be calculated as:

$$e_i = n_i - \hat{n}_i, \quad pe_i = \frac{n_i - \hat{n}_i}{n_i}$$

Then we use the median of the absolute errors, the median of the absolute percentage errors, and the average rank as the evaluation metrics.

$$MdAE = \text{median}\{|e_i|\}, \quad MdAPE = \text{median}\{|pe_i|\}, \quad AVGR = r_i$$

### E Augmented and Hard Test Sets in Sequence Labeling

The augmented test set is created by reasonably perturbing the numerals in a sentence. For example, for a numeral ‘173’ that describes height, we generate new samples by changing ‘173’ to ‘174’ or ‘175’ while keeping the other non-numerical words in the sentence unchanged. For a decimal such as ‘1.7 meters’, we change it to ‘1.6’ or ‘1.8’. The perturbation will not change the decimal places of numerals and will only change the quantity slightly, which makes the generated sentences reasonable.

The hard test set is created by manually collect ‘hard’ samples in the original test set. Hard samples do not have explicit patterns, meaning that a numeral’s tag cannot be easily inferred by its adjacent words. For example, tags of numerals followed by units like ‘cm’, ‘m’, ‘kg’, ‘years’ and ‘feet’ can be figured out easily, so we exclude them from the hard test set. Customers are very likely to use ambiguous expressions like: ‘I’m 16.5, can I buy 24?’, where 16.5 is about foot length and 24 is the shoe size. These ambiguous sentences are included in the hard test set.

### F Statistics of Sequence Labeling Dataset

We show the statistics of the customer-service dataset in the Table 10. The vocabulary is small because the dataset is confined to a specific domain: online customer service chat log about apparel purchase. In this dataset, most of the sentences are about sizes of various kinds of clothes and are very short and ambiguous.