OceanText: Visual Embedding of Chinese
Interim Report – COMP4801 Final Year Project

Yao Qingning, UID: 3035238644       Shen Zhuoran, UID: 3035233228

January 17, 2019

Supervisor:
Dr. Kenneth K.Y. Wong
Department of Computer Science
The University of Hong Kong
Abstract

Embedding is a foundational technique in modern natural language processing. It is necessary to convert text to vectors before feeding to neural networks for further processing. However, current approaches are predominantly developed on alphabetical languages, where individual characters have no meaning, and the meaning and the appearance of text are unrelated. In contrast, Chinese is logographic. Each character in Chinese is meaningful, and the meaning is correlated to its appearance. Therefore, this project proposes OceanText, a visual embedding system of Chinese, which utilizes the visual features of Chinese characters to improve the accuracy of embedding. The project team has finished literature review, implemented an embedding library as the baseline code base, and written a preprocessing library to automate data preprocessing. In the remainder of the project, the team will open-source the embedding and preprocessing libraries and implement and evaluate the OceanText algorithm.
## Contents

1 Introduction ........................................... 1

2 Related Works ........................................ 2
   2.1 Generic Word Embedding .......................... 2
   2.2 Chinese Embedding ............................... 3
   2.3 Visual Embedding of Chinese ................. 3

3 Method ............................................. 4
   3.1 Generic Formulation of Language Embedding .... 4
   3.2 The OceanText Algorithm ....................... 5
   3.3 Implementation .................................... 6

4 Current Progress .................................... 6

5 Project Progress .................................... 7
   5.1 Data Preparation ................................... 7
   5.2 Implementation .................................... 8
   5.3 Evaluation Methodology ......................... 8

6 Conclusion .......................................... 8

References
1 Introduction

Deep learning has been a central technique for recent advancements in natural language processing (NLP). The mathematical form of deep neural networks requires text to be converted to vector representations before being fed into the networks. This process is language embedding. As a necessary step for any deep learning-based NLP system, language embedding has been a foundational technique in modern NLP. The quality of embeddings has a significant impact on all subsequent systems that use them. For example, Artzi, Kwiatkowski, and Berant (2014) demonstrated the application of word embeddings in the context of semantic parsing\(^1\), Google’s famous paper on Neural Machine Translation is also heavily based on word embedding (Wu et al. 2016).

![Figure 1: Example of the compositional property of Chinese radicals](image)

However, existing approaches have been predominantly developed on alphabetical languages like English. Chinese, as a language with a logographic writing system, has two fundamental differences compared to alphabetical languages. Firstly, a single Chinese character is associated with meaning, unlike letters in alphabetical languages. For instance, 火 (huǒ) means fire, 人 (rén) means person, and 电 (diàn) means electricity. All three examples are individual characters isolated from any context, but they retain their meanings. In comparison, the English letter l, when isolated from any context, is not associated with any meaning. Secondly, the appearance of a Chinese character is related to the meaning it expresses. Figure 1 shows an example of the composition of the meaning of radicals. These distinct properties indicate a potential to incorporate the appearance information of Chinese characters into language embedders.

\(^1\)The task of converting natural languages to a logical form, see Artzi, Kwiatkowski, and Berant (2014).
2 Related Works

2.1 Generic Word Embedding

Word embedding is the process of converting each word in a piece of text to a vector representation. *word2vec* (Mikolov et al., 2013) was the first successful embedding algorithm that enabled the currently predominant deep learning-based NLP techniques. It represents each word by a one-hot vector and uses a 2-layer fully-connected feedforward neural network to predict the appearance context words around it. They defined a context word as a word within a fixed-sized window around the target word. However, since for every word in the corpus, every possible word not present in the context window qualifies as a negative example, the number of negative examples is prohibitively large. The authors proposed a negative example sampling technique called noisy contrastive estimation to mitigate this.

*fastText* (Bojanowski et al. 2016) improved upon word2vec by incorporating sub-word information into the embedding process. The authors observed that words in alphabetical languages like English consists of recurring and semantic character sequences, known as word roots or morphs. For illustration, the English word *representation* consists of morphs *rel*, *pre-*, *-s*, *-ent*, and *-ation*. They utilize the recurrence and semanticity of morphs by extracting all character n-grams with \( n \) in a specified range from each word and represent a word by an n-hot vector indicating both its ID and its character n-grams before feeding into the network.
2.2 Chinese Embedding

Chen et al. (2015) observed that individual Chinese characters are semantic, unlike the letters in alphabetical languages. Hence, they proposed to utilize the semanticity of individual Chinese characters by jointly training a character and a word embedder. Similarly to fastText’s (Bojanowski et al. 2016) utilization of letter n-grams, CWE represents each Chinese word by an n-hot vector indicating both the IDs of its constituent characters and its own ID and feeds this into the embedding network.

However, there are over 50,000 distinct Chinese characters, far exceeding the total number of English or Latin letters. This results in the extreme rarity of the least common characters. The contextual supervision provided by even a large corpus might be insufficient for training effective representations for those characters. To remedy this, the JWE (Yu et al., 2017) manually specified a set of rules to decompose each Chinese character to a set of common sub-character constituents. The reduced sparsity ensures enough supervision signals for all sub-character constituents.

Despite the reduced sparsity, the decomposition rules in the JWE (Yu et al., 2017) is manually designed and might not be optimal. Cao et al. (2018) used a learning approach to improve the decomposition of Chinese characters. Their method classifies each stroke into one of the five coarse-grained categories and represents each Chinese character as a set of stroke n-grams, for n ranging from 3 to 12. The authors empirically tested their approach and achieved record accuracies on word similarity, word analogy, text classification, and named entity recognition in Chinese. However, the stroke sequence lacks information about the variations in the realization of a stroke.

2.3 Visual Embedding of Chinese

The GWE (Su & Lee, 2017) attempted to augment Chinese embedders with visual features. The authors trained a 10-layer convolutional autoencoder using rendered images of characters that produces vectors of a fixed dimensionality. When embedding a word, the model obtains a vector of the same dimensionality from a conventional word embedder. Then, the model extracts a visual feature vector from the image of each constituent character. Finally, the model computes the output embedding as the sum of the conventional embedding and all visual features.

The bitmap-enhanced embedder (Costa-Jussà et al., 2017) uses the flattened vector of the pixels of the image of a character as its visual feature. The method then concatenates the visual feature with a conventional embedding and then returns the concatenated vector. The authors
validated their approach on Spanish-Chinese translation. The model outperformed previous neural translators but was inferior to the most accurate phrase-based models.

Unlike the GWE (Su & Lee, 2017) and the bitmap-enhanced embedder (Costa-Jussà et al., 2017), The ID+CNN embedder (Dai & Cai, 2017) is a character embedder. The authors replaced the 2-layer feedforward network in a conventional embedder by a visual architecture. The architecture consists of 1 or 2 convolutional layers followed by 2 fully connected layers. It takes in the image of a character and outputs an embedding of it. By only substituting the neural network in the embedding pipeline, the model can utilize contextual self-supervision like conventional embedders. The authors trained the networks on the Chinese Gigaword dataset. Then, they conducted evaluation on the same dataset for language modeling and on the MSR and the PKU datasets for word segmentation. Since the Chinese Gigaword dataset is proprietary and close-sourced, it is not possible to reproduce their work.

3 Method

3.1 Generic Formulation of Language Embedding

Figure 3: Flow chart demonstrating the three steps of embedding for word2vec

This section presents a generic formulation of language embedding. Language embedding is a mapping from the space of a type of linguistic units (e.g. characters, words, sentences, etc.) to a vector space. It generally consists of 3 steps, namely encoding, embedding, and supervision.

Figure 3 illustrates the 3 steps for word2vec. The encoding process of word2vec transforms a word into a one-hot vector indicating the ID of the word via a dictionary look-up. Then, the embedding process projects that one-hot vector to an embedding vector by a linear layer. Finally, the supervision module predicts context words using a softmax classifier and computes a cross
entropy loss as the supervision signal. Then, the module computes the gradients for parameters in the model with respect to the loss using the backpropagation algorithm (Linnainmaa, 1970) and performs a parameter update using an optimizer (e.g. SGD (Robins & Monroe, 1985) or Adam (Kingma & Ba, 2015)).

### 3.2 The OceanText Algorithm

The proposed method is shown in Figure 4. The encoding process renders a character into an image. The image is then used as input to the embedding process. The embedding module consists of a convolutional neural network and a linear layer. The linear layer has the same capacity as the one in a conventional embedder and ensures that the embedding module possesses at least the representational power of a conventional linear embedding module. Lastly, the supervision module is not modified compared to word2vec for a fair comparison. Table 1 presents a detailed comparison of OceanText and existing methods.

Figure 4: Flow chart demonstrating the three steps of embedding for OceanText

<table>
<thead>
<tr>
<th>Feature \ Method</th>
<th>word2vec</th>
<th>CWE</th>
<th>JWE</th>
<th>GWE</th>
<th>cw2vec</th>
<th>BEE</th>
<th>ID+CNN</th>
<th>OceanText</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual information</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Character semanticity</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Character structure</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Image inputs</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>End-to-end trainable</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Contextual self-supervision</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Latest vision architectures</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Feature \ Method</td>
<td>word2vec</td>
<td>CWE</td>
<td>JWE</td>
<td>GWE</td>
<td>cw2vec</td>
<td>BEE</td>
<td>ID+CNN</td>
<td>OceanText</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>--------</td>
<td>------</td>
<td>--------</td>
<td>-----------</td>
</tr>
<tr>
<td>Widely-used large datasets</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes²</td>
</tr>
</tbody>
</table>

### 3.3 Implementation

The project team is currently in progress of implementing OceanText algorithm in PyTorch. PyTorch is an open-source library for heterogeneous scientific computing and deep learning. It provides a diversity of popular operations supporting deep learning algorithms, including convolutions, linear layers, long short-term memory cells, and softmax operations, which are important constituents of OceanText. Additionally, it provides automatic differentiation, which alleviates the need to implement the tedious and error-prone differentiation and backpropagation mechanisms. Compared to competing deep learning libraries like TensorFlow, MXNet, Caffe 2, and CNTK, PyTorch has two additional advantages. First, it features a simple API and intuitive semantics tightly integrated with native operations in the Python programming language. This creates a smooth learning curve assuming prior knowledge of Python. Furthermore, it uses dynamic computation graphs, i.e. it dynamically generates the graph depicting the computational relations between different variables when computing them. This allows flexible debugging and expedites research explorations.

### 4 Current Progress

The project team decided to build its own code base upon existing an open-source word embedding library in PyTorch. This approach would minimize the efforts spent to produce a working baseline that OceanText can build upon and compare with. Among openly available libraries that satisfy the needs, `pytorch_word2vec` has the largest number of users and the most active community. The active community endorses its correctness and effectiveness. Therefore, the team chose it as the basis of the OceanText code base.

However, `pytorch_word2vec` has two major deficiencies. First, its speed is a major concern. Using the Wikipedia Chinese corpus, the standard dataset for Chineses NLP, it would take 82 days to train a single model. This speed is significantly slower than the theoretical calculation and unaffordable for research purposes. Additionally, the code quality of the library was poor.

---

²OceanText will be validated on widely-used large-scale datasets. However, the experiments have not be conducted yet.
It featured copy-and-pasted code, commented-out lines, oversized classes and functions, and other symptoms indicating unsatisfactory code quality.

To improve pytorch_word2vec and make it practical for research purposes, the team decided to refactor it to improve its code quality, since high-quality code is more analyzable, modifiable, testable, and maintainable. After the refactoring, the resultant code base offered the same functionality and performance with less than 300 lines of code, down from the over 1500 lines of the original code base. Furthermore, it eliminated all identified symptoms of poor code quality, including the ones mentioned in the previous paragraph.

With a high-quality code base, an investigation quickly pinpointed a defect that caused the significant slowdown of the library. After fixing the defect, the refactored code base runs over 70 times faster than the original. It now takes slightly more than a day to train a model, which makes the training of word embedder on widely-used large-scale corpora like Wikipedia Chinese viable. The resultant library is named OceanEmbedding. Table 2 summarizes the major differences between OceanEmbedding and pytorch_word2vec.

Table 2: Comparison between OceanEmbedding and pytorch_word2vec. Cyclomatic complexity is a widely-used metric to measure code complexity.

<table>
<thead>
<tr>
<th>Metric</th>
<th>OceanEmbedding</th>
<th>pytorch_word2vec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Time(^3)</td>
<td>28.1 hours</td>
<td>82 days</td>
</tr>
<tr>
<td>Logical lines of code</td>
<td>287</td>
<td>1568</td>
</tr>
<tr>
<td>Avg. cyclomatic complexity</td>
<td>2.54</td>
<td>3.76</td>
</tr>
</tbody>
</table>

5 Project Progress

5.1 Data Preparation

We have finished the data preparation tasks, including data source selection and the implementation of data pre-processing, cleaning, and loading scripts.

\(^3\)Single model on Wikipedia Chinese.
5.2 Implementation

Our plan was to implement the word2vec model, reproduce the existing results and implement our model based on that implementation. At this point, we have finished implementing the word2vec model in PyTorch, successfully reproduced the results and in progress of implementing our model.

5.3 Evaluation Methodology

We have devised a set of evaluation approaches. On the word level, we could use the existing WordSim datasets for checking the embedding’s accuracy of estimating word similarity, and on the character level, we could measure the performance gains in the downstream task of Chinese word segmentation.

使用中文的分词任务作为基准。

↓

使用中文的分词任务作为基准。

Figure 5: Chinese word segmentation example

6 Conclusion

In the past few months, we have finished literature review, data preparation, and model design. We are still in the process of implementing our designed model and we are close to reproducing the results of word2vec. For the next steps, we will continue with our implementation and optimization, then shift our focus to collecting evaluation metrics and publish our results.
References


Chen, Xinxiang, Lei Xu, Zhiyuan Liu, Maosong Sun, and Huan-Bo Luan. 2015. “Joint Learning of Character and Word Embeddings.” In IJCAI, 1236–42.


Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. “Distributed Representations of Words and Phrases and Their Compositionality.” In Advances in Neural

