Banki, a Chinese educational tool

Part 1: Vectors!

Edward Huynh

Abstract

This paper proposes a concept for leveraging word embeddings to teach semantic relationships in the Chinese language. I used the Chinese Wikipedia's data dump to create an initial test model. Then, based on a dataset consisting of Baidu Baike articles, I created a word2vec model which can identify implicit morphological and explicit semantic relations. I hope to combine this with a variation of the Anki algorithm to produce a powerful Chinese educational tool for students.

1. Introduction

So I was watching a 3B1B video about transformers the other day and I thought that this was really cool! What I felt was most thought-provoking was that the model was able to grasp semantic meaning in a way that was truly organic, with it having an almost human understanding of the vocabulary. So then I thought, 'Wouldn't this be really cool for Chinese?' And so here we are. For the non-Chinese people reading this, I'll leave an appendix at the end with all the translated words.

2. Preliminary research

I was foolish enough to think that people would just have a vector database for the Chinese language lying around, but no. A precursory search on Google yields promising results, for example this and this, but unfortunately, both require you to download from Baidu cloud, which is not gonna happen. I eventually stumbled on this Github repository which was surprising very informative, despite being 9 years out of date. As a result, most of the Python scripts were outdated and would not work, however the overall process was very similar to what I wanted to achieve.

After fixing up the scripts, I followed the example Wikipedia article dump to produce a model that was relatively good, considering it only took about 3 hours in total to make and train.

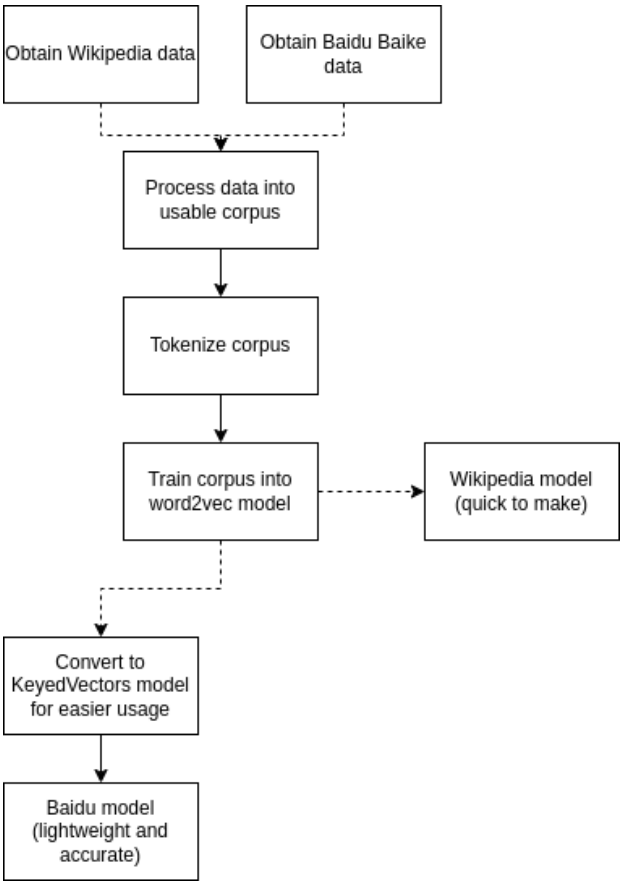

Figure 1: Flowchart of my approach, with the end goal being the a relatively accurate vector embedding model for the Chinese language.

```
>>> model.wv.most_similar('说')
[('说道', 0.6334753036499023), ('认为', 0.5915879607200623), ('答道', 0.586654782295227), ('告诉', 0.5718361139297485), ('指出', 0.5618188381195068), ('却说', 0.5551019310951233), ('讲', 0.5506150722503662), ('所说', 0.547660231590271), ('要说', 0.5415254831314087), ('时说', 0.5332564115524292)]
```

Figure 2: Testing the initial model's ability to find similar words.

```
>>> model.wv.most_similar(positive=['意大利人', '希特勒'], negative=['德国人'])
[('墨索里尼', 0.5258408784866333), ('佛朗哥', 0.4410151243209839), ('法西斯', 0.4274986684322357), ('加里波第', 0.42148908972740173), ('朱利奥', 0.4207106828689575), ('意大利', 0.417351573056732), ('法西斯党', 0.41183534264564514), ('马志尼', 0.40404263138771057), ('切萨雷', 0.39640527963638306), ('朱塞佩', 0.39441353082656860)]
```

Figure 3: Testing the initial model's ability to understand the semantics of vocabulary. It was able to recognise that Italian + Hitler - German → Mussolini, but with a low similarity (0.526). Comparisons will be made later on.

3. What's happening?

This paper covers my methodology to create a vector model for the Chinese language. But why? Why is this useful?

In essence, I'm trying to create a model which is able to understand Chinese in an authentic, natural way. Instead of having it simply a dictionary lookup, the goal is to have a model that *understands the relationships between words, not simply that the words exist.*

This whole project is based on a pretty basic idea from the NLP field (Natural language processing) which is that a dataset of text can have its 'tokens' stored in terms of vectors, which *embeds* their relationships between each other. For example, suppose this is the text I train my model on:

> "The Spring festival is my favourite festival. The Mid-Autumn festival is my favourite festival."

The hope is that this text can be used to create some model which recognises how 'Spring festival' and 'Mid-Autumn festival' are used similarly, and thus are similar terms. Furthermore, with a much more extensive dataset, the model should be able to understand these terms from a complex, semantic perspective. For example, the term 'Spring festival' should have the words 'Spring' and 'festival' embedded into its meaning.

This is where vectors come on. Essentially, with extremely high dimensional vectors, we can iterate over large texts of data, which are called corpora (singular is corpus) and construct a set of vectors from the vocabulary of the text, called tokens [1].

By iterating over our corpora, we are able to store the semantic meanings of the tokens as vectors. A simplified explanation of how this is done is that we are able to create base vectors that encode the fundamental meanings of things. For example, we could have a vector for hardness, and metallic-ness. From these fundamental vectors, we are able to build up our vocabulary through combining and varying the magnitude of these vectors. For example, the word 'wall' could be comprised of the vector for hardness with a high magnitude, and a small vector for metallic-ness, because 'wall' is not specific in reference to whether it's a steel wall, a stone wall, etc.

This wholly relies on the high-dimensionality

[1] Note that tokens aren't exactly the vocabulary, but that's a bit of a tangent, so if you're interested you can read this and do more research by yourself.

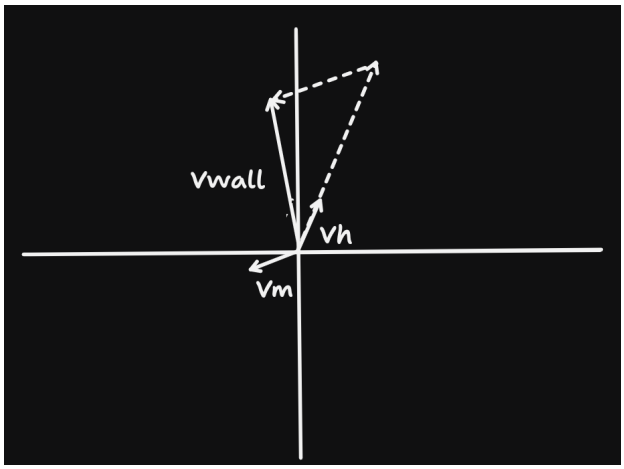

Figure 4: Extreme simplification of what a 'wall' vector may contain.

of the vectors. Because there are so many dimensions, we can encode semantic meanings on all these dimensions. For example, the vector for 'friendliness' could be stored as:

$$\mathbf{v}_{\text{fren}} = \begin{bmatrix} -3.1496480 \times 10^{-1} & -5.5083423 \times 10^{0} \\ -1.7922379 \times 10^{0} & -3.0600383 \times 10^{0} \\ -1.6571548 \times 10^{0} & 1.4378819 \times 10^{-1} \\ 1.9671721 \times 10^{0} & 5.8032078 \times 10^{-1} \\ 1.3521380 \times 10^{0} & -1.3395215 \times 10^{0} \\ -1.5511565 \times 10^{0} & 1.2028406 \times 10^{0} \\ \vdots & \vdots \end{bmatrix}$$

In summary, the goal is to analyse large sets of data and use them to create a vector model which encodes the semantic meanings of the dataset's vocabulary.

If you're still confused, please please please watch 3B1B's amazing video about GPT, specifically the section about word embeddings. He explains it much better than I can.

4. Processing Baidu corpus

The previously mentioned Wikipedia corpus was 2.5GB, however I was able to find a much larger corpus of Baidu Baike articles on Hugging Face. In comparison, this file was a whopping 16GB! However, this was formatted differently to the Wikipedia dump, hence I would have to modify the processing script before tokenization.

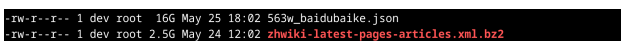

Figure 5: Size comparison of corpus

In hindsight, I probably shouldn't have included the 'Summary: ' and 'Section Text: ',

but I guess having a million of those wouldn't really hurt. However this definitely increases the size of the processed data file for no reason.

5. Tokenization

The previously mentioned Github repository suggested two methods of tokenization: either splitting each character individually, or by using Jieba to segment the text. I opted to use Jieba, however upon running the script I immediately encountered the classic 'OOM killer' error, because trying to process 2.5GB apparently overloaded the 8GB of memory I had available.

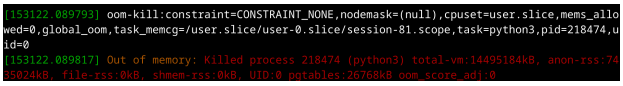

Figure 6: Certified "(t)OOM(wn)" moment

Instead, I had to use a workaround to this memory issue by processing the file in chunks.

Thankfully this worked quite well, however it made tokenizing the initial model take very long. I had to increase the chunk size from 1048 to 40960kB for decent times. **Note that a chunk size of 1048kB resulted in roughly a 27% utilisation of my RAM while processing a 2.5GB corpus. (This is for the Wikipedia corpus)**

6. Training

As mentioned previously, training the initial model was quite simple and quick, taking around 3 hours. However this was not the case for the Baidu Baike corpus. My estimations are that it took around 14 hours of continuous CPU usage to train, however the specifications for this model were quite over the top, as follows:

- Vector size=300
- Window=10
- Min count=20
- Epochs=10

In particular, using 10 epochs was a bit ridiculous in hindsight. It would have been fine with 5 epochs, however this did make the model a little bit more accurate. It may be useful to consider training with even more epochs over a longer period of time to further refine the model, but this results in a problem of power usage.

6.1. Power usage

My estimate for the power usage when training the model is a whopping $19.7W$ of power! Note that this was a semi-headless server, with it being bare-bones Debian. Over 14 hours, that should be about $0.0909955214 worth of computing time, which is probably wrong because I couldn't find a power to cost calculator online. This means running multiple epochs could get quite expensive, and far outweigh the returns in accuracy for the model.

7. Comparing models

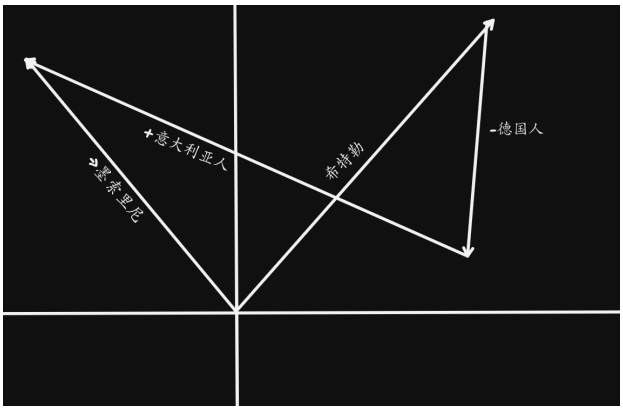

Figure 7: Visualisation of semantic relationship between '希特勒' and '墨索里尼'. Note that this diagram only represents 2-dimensional space, while both models were trained to make 300-dimensional vectors. In this case, the semantics of '意大利亚人' and '德国人' are able to be represented in high-dimensional vector space.

	意大利人 + 希特勒- 德国人 → 墨索里尼	说 → 告诉	春节 → 元旦
Wikipedia	0.52584	0.57184	0.52667
Baidu Baike	0.75288	0.63099	0.74542

Table 1: Comparison of similarity scores

The preliminary results show that compared to the smaller model trained on Wikipedia's data, using the Baidu Baike model yielded stronger results in terms of the model's understanding of the semantics in the examples. However, inspection of both model indicates that they primarily seem to be identifying similar words through vaguely similar usage, as shown below.

春节	新年	过年	清明节
朋友	好友	挚友	同学
你	我	🄵	我们
漂亮	可爱	帅气	脸蛋 *
化学	有机化学	chimie	分析化学

Table 2: Wikipedia model's top 3 most similar words, in decreasing order of similarity

春节	过年	新年	中秋节
朋友	好友	伙伴	老朋友
你	我	您	别人
漂亮	很漂亮	好看	可爱
化学	物理化学	物理	有机化学

Table 3: Baidu model's top 3 most similar words, in decreasing order of similarity

From inspecting the tables, it is clear that the Baidu model has a much firmer grasp on the semantic meanings of each word, and is more accurately able to identify synonyms. However, both models suffer from being unable to distinguish words that are used similarly, for example 春节 and 中秋节/清明节 are interpreted as having similar meanings, as they are used in almost exactly the same way.

Despite this, the Baidu model is more accurately able to identify synonyms in comparison to the Wikipedia model. For example, when finding similar words for ' 漂亮', the Baidu model accurately and confidently found correct synonyms, compared to the Wikipedia model which provided synonyms with less confidence and an additional erroneous synonym(脸蛋).

Oddly, for more uncommon vocabulary (e.g. 化学), the Wikipedia model seemed to be more accurate than the Baidu model (ignoring the erroneous English). This can be assumed to be because the processed Wikipedia data had obscure articles linked together, increasing the weighting of these similar terminologies. It should be noted however that the Wikipedia model was a lot less confident in these results. (0.722, 0.711, 0.694 for Baidu, 0.603, 0.589, 0.564 for Wikipedia) Furthermore, the much larger size of the Baidu corpus may mean that terms being used in a similar fashion are overrepresented as similar words. This may explain why ' 物理', which can be used similarly to ' 化学' appears in the Baidu model's results.

8. Conclusion

A precursory analysis indicates the Baidu model has a strong understanding of the semantic relationships of the Chinese language compared to the Wikipedia model, however more testing with more intricate examples is required. Before being used for the intended educational tool, the Baidu model should be more refined with further training.

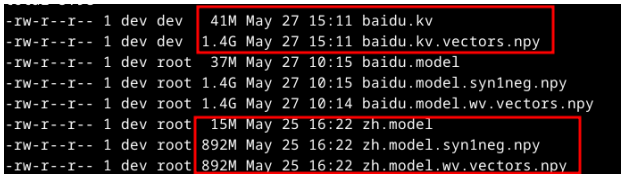

Figure 8: Comparison between model file sizes. Note that the Wikipedia model is the raw Word2Vec model, while the Baidu model has been converted to KeyedVectors, which are much faster to read. While the Wikipedia model is roughly the same size as its corpus, the reduced Baidu model is significantly smaller, at 1.4GB compared to 16GB. This suggests more repetitive data from the Baidu corpus.

Acknowledgements

The 3B1B video which inspired me to dig deeper. Candlewill's Github repository which started me on this journey. Gensim's word2vec and KeyedVectors documentation for helping me create the vector embedding models. Wikipedia for graciously making their data dumps available. Qinyang Xu for providing the dataset of Baidu Baike.

A. Appendix 1: Translation table

Translations thanks to the CC-CEDICT dictionary.

Chinese (中文)	**English (英语)**
说	to speak; to talk; to say
告诉	to tell; to inform; to let know
春节	Spring Festival (Chinese New Year)
元旦	New Year's Day
新年	New Year
过年	to celebrate the Chinese New Year
清明节	Tomb Sweeping Day
朋友	friend
挚友	intimate friend
同学	classmate
你	you
我	I; me; my
我们	we; us; ourselves; our
漂亮	pretty; beautiful
很漂亮	very beautiful
好看	good-looking
可爱	cute
化学	chemistry
物理化学	physical chemistry
物理	physics
有机化学	organic chemistry
伙伴	partner
老朋友	old friend
中秋节	Mid-Autumn Festival
您	you
别人	other people; others; other person
帅气	handsome
脸蛋	cheek, face
分析化学	analytical chemistry
意大利人	Italian person
希特勒	Adolf Hitler (1889-1945)
德国人	German person or people
墨索里尼	Benito Mussolini (1883-1945)