Semantic Analysis and Sentence Difference using TensorFlow

Simon Yip
CSCI 49362
Language Technology
Prof. Jia Xu

What is Semantic Analysis

- The process of relating syntactic structures from the phrases, clauses, sentences, and paragraphs to the level of the writing as a whole, to their language independent meanings.
- 2 Types of Semantic Analysis are Semantic Role Labeling (SRL) and Latent Semantic Analysis (LSA).
- Semantic Role Labeling uses the semantic arguments of a text using its predicates or verbs to classify other words into their specific roles.

Latent Semantic Analysis

- LSA is the analysis of relationships betweens the texts and terms they contain by using the concepts that connect them. It uses the idea that words similar in meaning will appear in similar texts.
- The LSA approach plots words based on how often they appear in a text on a matrix and uses Singular Value Decomposition to condense the matrix.
- The cosine similarity of the vectors formed by any 2 rows in the matrix is then taken to represent how close the words are.

What is TensorFlow

- TensorFlow is Google's open source software library used for numerical computation using data flow graphs.
- The nodes in the graph represent mathematical operations and the edges of the graph represent multidimensional arrays called tensors, communicated between the arrays.
- In this project, I used the Word2Vec implementation provided on the TensorFlow website and the GPU version of TensorFlow to represent words as vectors.

How does Word2Vec work

- Vector Space Models are used to represent embedded words in a vector space. Words that are semantically similar are mapped near each other
- This idea is based on the distributional hypothesis which says that words used in a similar context are semantically related
- The frequency of which words occur with its neighbor in a corpus is recorded using count based models and mapped to a small vector.
- My implementation of Word2Vec uses a training corpus of about 5,000,000 words.

Skip-gram Model

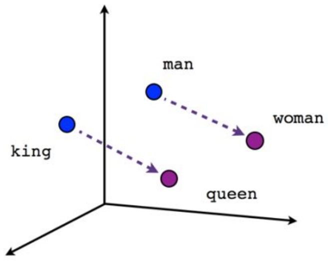
- Word2Vec uses the Skip-gram model which attempts to find the source words from a target word. So given the phrase “The cat eats”, a Skip-gram model tries to predict “the” and “cat” from “eats”, “cat” and “eats” from “the”, and “the” and “eats” from “cat”
- Then Stochastic gradient descent or SGD, is used to determine the “loss” between the input and output of the Skip-gram model and the noisy data.

Skip-gram Model (cont.)

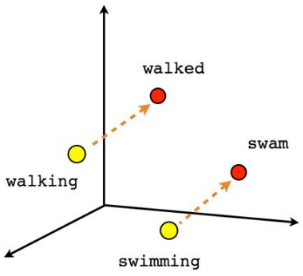
- Using TensorFlow's helper functions, we can derive the gradient of the loss with respect to the embedded parameters and move the word embeddings closer to their semantic meanings.
- This is done many times when training the training set. Words are repositioned to differentiate meaningful words from noisy words.

Relations between Vectors

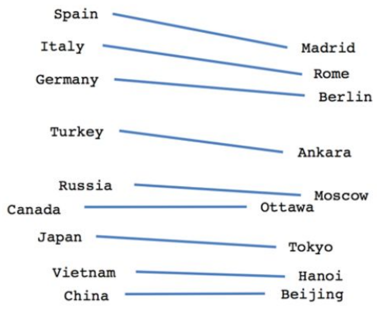
- From observing similar words, it becomes apparent that some word pairs that are semantically related share similar directions.
- For example, man is semantically related to woman in a similar way as king is semantically related to queen.



Male-Female



Verb tense



Country-Capital

Creating the Graph

- To construct the graph of word embeddings, we need to define and use an embedding matrix.
- The estimated loss is found using a logistic regression model with weights and biases for parameters.
- The Skip-gram model takes a list of integers that represent the source words as well as a list of target words (which are created during preprocessing).

Creating the Graph (cont.)

- The Word2Vec implementation then finds the vector for each source word by using TensorFlow's helper functions.
- It then predicts the word using the noise-contrastive training objective to find the loss node.
- Once we have the loss nodes, we can use the aforementioned Stochastic gradient descent (SGD) to optimize parameters and calculate gradients.

Training the Skip-gram Model

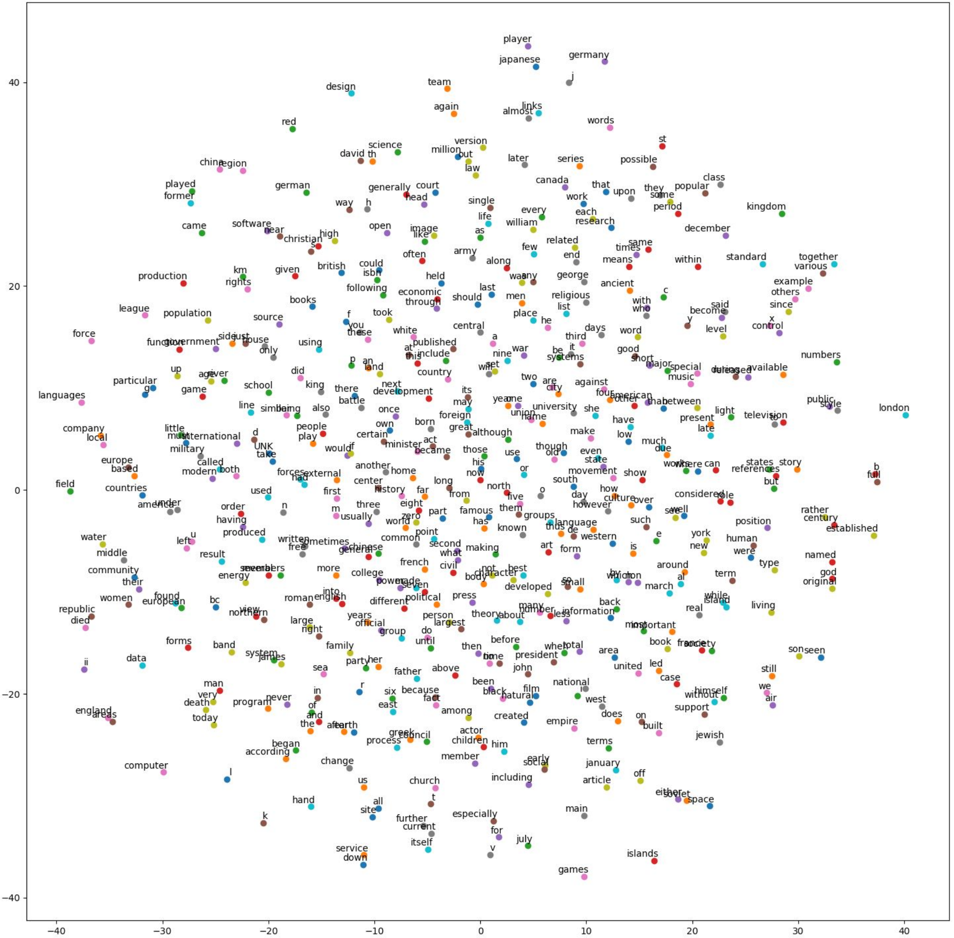
- The data is trained in steps, the more steps we use, the less we lose and the closer we get to an accurate representation of the word.
- The Word2Vec implementation uses 100,000 steps but shows diminishing returns every 10,000 steps with negligible improvements after around 70,000 steps as shown in the next slide

```
Average loss at step  0 :  313.963012695
Average loss at step  2000 :  148.789294937
Average loss at step  4000 :  86.6500731459
Average loss at step  6000 :  62.2829870286
Average loss at step  8000 :  48.1217757032
Average loss at step  10000 :  39.5772818255
Average loss at step  12000 :  31.7750077488
Average loss at step  14000 :  27.5785483205
Average loss at step  16000 :  23.3059112493
Average loss at step  18000 :  19.8172764577
Average loss at step  20000 :  17.7868184454
Average loss at step  22000 :  15.2312061284
Average loss at step  24000 :  14.248663885
Average loss at step  26000 :  13.6222507396
Average loss at step  28000 :  12.1144339666
Average loss at step  30000 :  10.7962964731
Average loss at step  32000 :  10.2988698584
Average loss at step  34000 :  9.50459757912
Average loss at step  36000 :  9.43660793674
Average loss at step  38000 :  8.7354097544
Average loss at step  40000 :  7.92184068584
Average loss at step  42000 :  8.04514906991
Average loss at step  44000 :  7.30438592517
Average loss at step  46000 :  7.3575727427
Average loss at step  48000 :  7.33983593214
Average loss at step  50000 :  6.65150517511
```

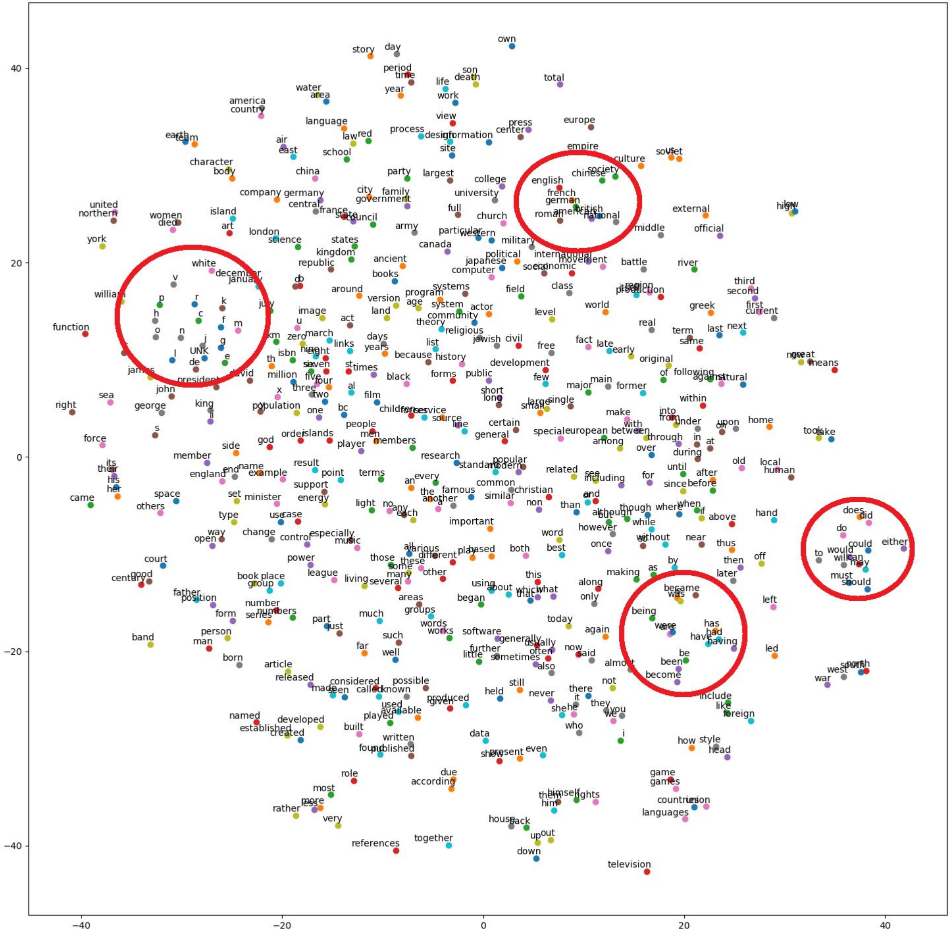
```
Average loss at step  52000 :  6.63962312651
Average loss at step  54000 :  6.91327071965
Average loss at step  56000 :  6.49968332517
Average loss at step  58000 :  6.52918291557
Average loss at step  60000 :  6.16855059159
Average loss at step  62000 :  6.29052153575
Average loss at step  64000 :  6.0500851416
Average loss at step  66000 :  5.60809822595
Average loss at step  68000 :  6.24757198143
Average loss at step  70000 :  6.04111240602
Average loss at step  72000 :  5.73375457096
Average loss at step  74000 :  5.80888203609
Average loss at step  76000 :  5.66138692498
Average loss at step  78000 :  5.98879583162
Average loss at step  80000 :  5.99030760467
Average loss at step  82000 :  5.75701002192
Average loss at step  84000 :  5.67182213926
Average loss at step  86000 :  5.71052172613
Average loss at step  88000 :  5.65827141082
Average loss at step  90000 :  5.56746507108
Average loss at step  92000 :  5.29869867313
Average loss at step  94000 :  5.5707445215
Average loss at step  96000 :  5.42998508513
Average loss at step  98000 :  5.20181859219
Average loss at step  100000 :  5.56816608226
```

More on Accuracy and Steps

- The next slide will have 2 graphs plotted after training with different numbers of steps. The data on the left is the graph after 10,000 steps while the data on the right is the graph after 100,000 steps.
- The graph on the left shows the data without significant meaning. This is because the data has not been adequately trained yet. The words seem randomly placed on the plot whereas the graph on the right depicts some semantically related words closer to each other.
- For example you will see languages, words with the same part of speech, and letters grouped together.



40
20
0
−20
−40
−40 −30 −20 −10 0 10 20 30 40



40
20
0
−20
−40
−40 −20 0 20 40

Cosine Difference of Sentences

- Cosine difference, also known as dot product, is a method of measuring the angle between vectors.
- It can be used to measure the difference (or similarity) of 2 sentences given their vectors.
- This is similar to how edit distance measures the difference between sentences. Cosine difference on the other hand, should be able to measure the difference on a deeper level in terms of semantics.

How to Find the Cosine Difference of 2 Sentences

1) First, we need to preprocess the 2 sentences by changing them to lower case and splitting the words into vectors.
2) We need 2 vectors of size *n* where *n* is the number of words in the longer sentence.
3) We then populate these vectors with the distance from 1 word embedding to the next. For example, the length of the 1st word to the 2nd, the length of the 2nd word to the 3rd and so on until the last word in the sentence.
4) Finally, calculate the cosine difference of the vectors using one of Python’s various math libraries.

How to Populate Sentence Vectors

- We use the Pythagorean Theorem to find the distance between two word embeddings.
- If the word is not in the dictionary, we assign it a coordinate of 0,0 to stabilize the impact it will have on the result by making the unknown word equidistant from all other words on average.
- If one sentence is shorter, we can fill up the vector with lengths of 0 until both vectors are the same size. This is because we cannot find the cosine difference of 2 vectors of different size.

An Example of Cosine Difference of Sentences

Find the cosine difference of the following 2 sentences:

the human built the house

the german people designed that house together

```
['the', 'human', 'built', 'the', 'house']
['the', 'german', 'people', 'designed', 'that', 'house', 'together']
[13.330700447827637, 24.544139491857763, 31.635678940059464, 41.662908313473174, 0, 0]
[26.306717567726764, 55.118108627266565, 32.471217523681204, 18.38089909155693, 42.701593283375686, 39.03337713670877]
0.359310808411
```

The cosine difference is .359310808411

Another Example

Now let's try a more syntactically similar set:

George was born in China on July second

On January first David was born in America

```
['george', 'was', 'born', 'in', 'china', 'on', 'july', 'second']
['on', 'january', 'first', 'david', 'was', 'born', 'in', 'america']
[28.998256019927922, 9.11554733917285, 21.94568401380408, 43.27062567981256, 48.54354707657445, 20.672835560824442, 25.783883
864660474]
[20.543796929618402, 27.190878803464685, 32.54102151281813, 31.718747500384538, 9.11554733917285, 21.94568401380408, 44.39422
927104563]
0.20151596379
```

The cosine difference is .20151596379

Edit Distance

the human built the house

the german people designed that house together

Edit Distance = 5

George was born in China on July second

On January first David was born in America

Edit Distance = 8

Comparing Edit Distance to Cosine Difference for Set 1

First set:

Edit Distance of 5 out of 7 words

Cosine Difference of .3859

The edit distance says that you need to change over half the words in the sentence to make the other. In terms of meaning, the sentences are quite similar with “human” and “german people” being the nouns and the verb “design” is close to “build”. In this case the cosine difference is telling us that the sentences are closer in meaning than they appear.

Comparing Edit Distance to Cosine Difference for Set 2

Second Set:

Edit Distance of 8 out of 8 words

Cosine Difference of .2015

The edit distance tells us that the sentences appear to be drastically different. The structural differences are the dates, the name, the place, and the order of the sentence. The Word2Vec model can tell that all 3 sets of fields are related and the cosine difference tells us that the two sentences are very similar in meaning. Thus, the cosine difference is better at finding sentence difference in terms of semantic meaning.

Things I learned and Possible Improvements

- Machine learning is actually very random in that the Operating System executes different threads of the program in different orders and different speeds. Every time the training occurs, it will result in a different model regardless of the corpus or parameters.
- There are many different Training Texts available to use. I used the default corpus provided by Word2Vec, but noticed a lot of noisy words and a lack of some commonly used verbs like “run” or “walk”. Trying different Training Texts could have made my implementation more extensive and accurate.

Works Cited

https://www.tensorflow.org/tutorials/word2vec

www.google.com for Python syntax, functions and general definitions of terms

Class notes