Progress report

Abstract

At this stage of the experiment, we utilized a model of English words, represented by vectors, to analyze relevance of long articles and short sentences. We simplified input articles and sentences using frequency counts of each word's appearances, and calculate the similarity between words. By using vector representation of each word, we are able to take into account semantically similar words (such as university/universities, buy/bought, cat/kitten), which yield significantly more accurate results, especially in shorter sentences, and highly relevant articles.

Introduction

The main challenge of semantic-level analyses of articles ranges from the sparsity problem, the fact that appearance of each English words are not frequent enough), to the knowledge representation of semantics, the fact the computer cannot directly detect the similarity between words other than identical string tokens. (For example, singular-plural forms, grammatical tenses, and lexically similar words are accounted as different words.) The consequence is, it is not viable to analyze sentences or questions that are of lower order of magnitudes than entire English lexicons. (The older method almost always return 0 similarity, since words are often all different.) We tackle the problem by representing each word as a high-dimensional vector, where we can directly calculate semantic similarity between words. It turns out that semantically similar words, regardless of grammatical forms or small deviations in the lexical meaning, generally fall into the similar vector values. The model also captures the second order meaning (2x2 word analogy), where it maps the words such as "go-went : buy-bought", "king-queen : male : female" on to a parallelogram. Using this model, we can account for a similarity between

words by calculating the similarity between each word as well. This mitigates the sparsity problem arose in the extremely short prompt (such as sentences or questions).

Methods

Step 1: Obtain the training set

Our training set is the text8.zip file from http://mattmahoney.net/dc/, which is the resource that Google releases for creating the vector representation of words on the latest research tool *TensorFlow*^{TM}. This file contains many articles, having in total above 17 million words, which is sufficient to create a vector representation of almost all English words.

Step 2: Training Algorithm

We apply the gensim library, which internally applies the deep learning algorithm, especially hierarchical softmax skip-gram algorithm. (This library uses the similar algorithm to *TensorFlow*^{TM} ; however, gensim library has the better interface for training and storing data.) We use this algorithm because the hierarchical softmax algorithm can deal with the great number of inputs with the small computational complexity per training instances, says, O(log V) where V is the number of the word-output vectors. Since gensim library requires having an input as a matrix of words where each row represents a sentence and each column represents a word in the sentences, we apply the simplified model by separating our training data into a sentence of exactly 10 words so that we have 1.7 million training sets. After passing the training set to the gensim function, we receive an output as the 128-th dimensional vector representation of each word and subsequently save such data in the separate file.

Step 3: Testing Algorithm

In both previous and new versions, we count the number of each word in each article stored in collections.Counter.

Previous version: (This version does not have the training algorithm.) We determine the cosine value of the angle between two vectors by calculating the dot products of two vectors (or counters), divided by the product by the L2-norm of both vectors. The greater the cosine value is, the more relevant between both articles.

Current version: We currently have two different algorithms to determine the relevance between two articles, which is better off in different situations (We will delve into more details in the following section.)

1st Algorithm: We iterate through all combinations of pairs of word where one word is in the counter of the first article and another word is in the counter of the second article. Then, we calculate the summation of the product of occurrences of both words and the similarity relevance we have from the model (where we can compute similarity($word_1$, $word_2$) from the model obtained from the training data). Then we normalize the value we obtain by dividing the product of L1-norm for both articles.

Pseudocode:

```
relevance = 0
for word1 in counter1:
     for word2 in counter2:
          relevance += counter1[word1]*counter2[word2]*\
                         similarity(word1, word2)

relevance = relevance/(‖counter1‖_1×‖counter2‖_1)
```

2nd Algorithm: This algorithm is similar to the first version, but the only different is we will calculate the summation of the product of occurrences of both word only in case that

similarity($word_1$, $word_2$) is equal to and greater than the threshold (we currently use threshold = 0.5.)

Pseudocode:

```
relevance = 0
threshold = 0.5 // Can be other values
for word1 in counter1:
     for word2 in counter2:
          if similarity(word1, word2) >= threshold:
               relevance += counter1[word1]*\
                         counter2[word2]*\
                              similarity(word1, word2)

relevance = relevance/(‖counter1‖₁×‖counter2‖₁)
```

Preliminary Results

We ran the new algorithm and old algorithm on Wikipedia articles and short sentences. Results are summarized in the tables below.

Wikipedia Pages

Page Title		Similarity		
First Page	**Second Page**	**Old Algorithm**	**First Algorithm**	**Second Algorithm**
Thailand	Bangkok	1.00	1.00	1.00
Thailand	Country	0.48	1.03	0.97
New York City	Statue of Liberty	1.02	0.87	0.98
Japan	Sushi	0.63	0.63	0.73
Computer	Alan Turing	0.72	0.90	1.06
Pad Thai	Sushi	0.58	1.76	6.00
Stanford University	University	0.73	0.95	1.78
Electricity	Electric Car	1.1	1.46	1.45
Elephant	Cat	0.93	1.67	2.20
Petroleum	Diesel Engine	0.62	1.90	3.00
Logic	Set Theory	0.64	2.25	4.22
Bee	Metamorphosis	0.48	1.94	3.70
Paper	Tree	0.59	1.733	2.10
Thailand	Axiom	0.40	0.36	0.33
Bee	New York City	0.24	0.60	0.543
Car	Alan Turing	0.55	0.65	0.65

Bicycle	Tree	0.75	1.66	1.22

Green: words are relevant; words are proper nouns
Yellow: words are relevant; words are generic nouns
Red: words are irrelevant
Note that this categorization is evaluated by humans. The values in the table are normalized by the similarities between "Thailand" and "Bangkok" obtained from three different algorithms.

Short Sentences

Short Sentences		**Similarity**		
1st	**2nd**	**Old Algorithm**	**First Algorithm**	**Second Algorithm**
I love dog	Dogs are lovely	0	1.7	2.49
I love dog	This puppy is cute	0	0.98	1.66
I love dog	This kitten is cute	0	0.86	0
I love cat	This kitten is cute	0	0.82	1.66
I witnessed a car incident yesterday	My father works at an insurance company	0	0.36	0.60
I will go to the beach this weekend	I really need to take a vacation	1.0	1.0	1
I love Japanese people	I love Japan because it is a beautiful country	2.5	1.33	1.24
Donald Trump is great	I love Republican party	0	0.38	0
It is really hot here	I hate summer	0	0.155	0

The values in the table are normalized by the similarities between "I will go to the beach this weekend" and "I really need to take a vacation" obtained from three different algorithms.

We can see that when evaluating Wikipedia pages relevance, the two new algorithms perform better than the old one. I.e. they give a wider range of similarity value. In addition, while the old algorithm fails to recognize the similarity between two articles that do not seem very obvious, such as "Logic" and "Set Theory"; "Diesel Engine" and "Petroleum", the two new algorithms seem to be able to recognize the similarity well. This is because the two new algorithms use the knowledge that words can have similar meaning although they are not the same, while the old algorithm only look for the same words in both articles. It is also worth

noting that the two new algorithms seem to yield lower similarity value when comparing articles with proper noun title. This is possible because when we train word2vec, there are not many proper nouns in our training set.

The main difference between the first and second algorithm is that the first algorithm seems to produce similarity values that are less extreme, while the second algorithm sometimes give a really high similarity value. This is mainly because we set a threshold at 0.5 (which is quite high), and the second algorithm will count two words as similar when they are in fact very similar while the first algorithm will add the product of similarities of any two words, however low that may be.

When we test three algorithms against short sentences, we can see that the new algorithm outperform the old algorithm. When two sentences have no overlapping words, the old algorithm will clearly produce zero similarity. The new algorithms, however, are able to recognize similarity between words and therefore can produce appropriate non-zero similarity values.

Challenges and future work

There are several challenges that we can address in the next level as we optimize our methods.

1. Specific / proper nouns vs common nouns and high frequency words (stop words) filtering. We have actually implemented the filter for high frequency words (high frequency words are cut off as we analyze both articles). However, some articles are of the same material (such as Thailand-Bangkok), but are analyzed as low relevance. We need to find a proper way to weight these type of words to be heavier than more common nouns.

2. Training data using dictionary as a prior

It is possible that we might use dictionary definition as a source of training data, but use some different method as a “primordial” model, before we use other sources (such as Wikipedia article).

3. Try the same model, but on a sentence level

Our “article vectors” are currently based on frequency counts of words. However, we are considering training large number of sentences to make a simple program to map locus of word vectors to a higher dimensional vector that represent the whole articles.