COURSE PROJECT- CS365: ARTIFICIAL INTELLIGENCE

BILLION WORD IMPUTATION

Aayush Mudgal
mudgal@iitk.ac.in
12008

Shruti Bhargava
shrutib@iitk.ac.in
13671

April 19, 2015

Abstract

The project Billion Word Imputation is an interesting application of machine learning and AI techniques to language modeling and can find use in natural language understanding and speech recognition. It explores machine learning methods for sentence modeling to efficiently figure out the missing word and its location.

1 PROBLEM STATEMENT

The Billion word Imputation challenge is a live Kaggle challenge and a slight variation from the classical language modeling task. In this challenge, from each sentence in the test set, exactly one word has been removed randomly. The removed word is essentially neither the starting nor the end word. Each sentence ends with a period '.' , which is always the last word of any sentence. The billion word corpus provided by *Chelba et al* [1] The challenge is to insert the correct word at its location in the imputed test sentences. The detailed problem description is available at Link to Kaggle site.

2 MOTIVATION

The Billion Word Imputation is a challenging task which involves the knowledge and application of ideas from natural language processing, machine learning and sequential data. A lot of research is being going on in language modeling, but this problem attempts to take the language modeling field to a different arena. Let for example consider the sentence

"Michael described Sarah to a at the shelter ."

A normal human, with moderate knowledge of English language could identify that there is something wrong with the structure of the sentence and also that there is a word missing, with ease. He/She will be able to figure out that the structure of the sentence should be as follows:

"Michael described Sarah to a ***<missing-word>*** *at the shelter . "*

Consider another sentence, chosen from the test dataset of this challenge -

"He added that people should not mess with mother nature , and let sharks be ."

Most of the average English speaking people, won't be able to figure out the mistake in this sentence and may assume it to be correct. But, grammatically the above sentence is incorrect, *"and let **the** sharks be"* is the correct phrase. Such sentences are difficult to be figured out manually.
A solution to this challenge would thus enable the machine to perform this task with more efficiency than what most humans could do, making this project idea an appropriate one for this course on Artificial Intelligence. It requires the analysis of the existing modeling methodologies in order to answer a different and a more challenging task. Insights into this problem could also be beneficial in decoding partially deciphered manuscripts and may also find its use in speech recognition.

3 INTRODUCTION

Our approach to the problem involved the creation of a language model, that will provide a decisive score that a given imputed sentence is generated by the learned language model. A Modified N-Gram Model, was developed for this purpose The second approach that we tested was to develop a language model that will predict the possible word using the context information of its neighboring words. This involved code modification of the Continuous Bag of Words model of Word2Vec's [2] implementation in python provided by gensim.

The whole problem can be sub-divided as follows

- Identification of the location of the missing word
- Identification of the missing word

Both the above sub-problems, are challenging tasks in themselves. The challenge here is to solve them together.

4 RELATED WORK

A partially similar problem was introduced by Microsoft Research Centre - 'The Microsoft Research Sentence Completion Challenge'.As described by the Micrsoft Research website:

This challenge involves a set of sentences, each of which has four impostor sentences, in which a single (fixed) word in the original sentence has been replaced by an impostor word with similar occurrence statistics. For each sentence the task is then to determine which of the five choices for that word is the correct one. It therefore requires a model to choose the best fitting word out of the given 4.

The KaggleâĂŹs challenge is different and seemingly a more difficult modification of Microsoft's sentence completion challenge, as herein we have to predict both the missing word and its location.
Cantab, the team currently ranked four in the competition proposed a model wherein they developed a model that is a combination of a well trained huge recurrent neural network and a KN 5-gram, details of which are found at (http://code.google.com/p/1-billion-word-language-modeling-benchmark/ and their ICASSP paper [3]. They developed a language model, that contained all words in all possible locations and asked the model to return the 100 most likely sentences, each of which has a corresponding likelihood.

If the posterior of the top one is greater than half, they use that sentence, else they do nothing. Their model takes a huge amount of time, in magnitude of weeks, to train and subsequently generate responses to the test set. Cantab's approach to the problem is specified at http://www.kaggle.com/c/billion-word-imputation/forums/t/12977/cantab-s-approach. We planned to solve this challenge using the nominal available system specifications, in a reduced time.

5 TRAINING AND TEST CORPUS

The corpus used for the competition uses the billion-word benchmark corpus provided by Chelba et al. [1] for language modeling. The data for this competition is a large corpus of complete English sentences.

- The training set consists 4.2 GB data of complete English sentences.
- The test set contains a number of sentences where one word has been removed randomly. The removed word is anything, except the first and the last word of the sentence.

6 METHODOLOGY

Two different approaches were proposed to solve this challenge and are discussed in further sections

6.1 LANGUAGE MODELS

The understanding of natural languages is very complex. Words essentially combine in a non-random order but in some complex order. It is intuitively believed that the, language models can be learnt from the word and its neighboring words (which define the context surrounding the word). This structure of languages could be exploited for sentence competition.

6.1.1 COMPLETE-VOCAB-SIMPLE N-GRAM MODEL

N-gram features are known to be capable of capturing the domain and syntactic knowledge about the sentence to a large extent. Consider a sentence S, from the test corpus, where S = Michael described Sarah to a at the shelter. The sentence S has a unknown missing word at an unknown location. To model the sentence, consider blank spaces (denoted by a _) at each of the possible locations iteratitively, i.e. consider the following set of sentences.

Michael _ described Sarah to a at the shelter .
Michael described _ Sarah to a at the shelter .
Michael described Sarah _ to a at the shelter .
Michael described Sarah to _ a at the shelter .
Michael described Sarah to a _ at the shelter .
Michael described Sarah to a at _ the shelter .
Michael described Sarah to a at the _ shelter .
Michael described Sarah to a at the shelter_ .

Each of the possible location in the sentence is a valid position for the missing word to be at. Each of the word in the trained vocabulary is a possible candidate for the missing location. The score of the generated sentence, i.e. the sentence probability to occur in the training dataset, with the missing word replaced by

some word of the vocabulary is calculated. This approach was tested considering a 5-gram model of words. Let $S = (w_1, w_2, ..., w_{n-1}, w_n)$ be an English sentence where $w_1 ... w_n$ are valid tokens. A sub-string of the sentence from index i to index j is denoted as $S_{i:j}$, such that $S_{i:j} \equiv w_i w_{i+1} ...., w_{j-1}, w j$
Let the missing word in the sentence is w_j at the location $i+1$ in the sentence S. Thus the task is modeled mathematically as:

$$S^{'} = S_{0:i} w_j S_{i+1:n}$$

$$Target\ sentence \equiv \underset{i}{\operatorname{argmax}} \prod_{k=i}^{n} P(w_k | w^{m-1})$$

The language model comprised of a 5-gram trained model using the KenLM toolkit. KenLM stands for Faster and Smaller Language Model Queries. The KenLM library implements two data structures that allow very efficient language queries. The KenLM library packs ways to use the Kneser-Ney smoothing method as a part of the KenLM. Knesser-Ney Smoothing is helpful for generalization and for testing, as not every possible n-grams could be captured in the training data. It helps in generalization and better results when the test data has words not already present in the vocabulary. Considering the same test sentence S as Michael described Sarah to a at the shelter, assume that each location can be a probable missing location, the score of sentence is found out as follows, where S_t is a probable sentence after the missing location has been fixed.

$$Score(S^t) \equiv \sum_{i=5}^{n-1} log(P(w_i | w_{i-1}, w_{i-2}, w_{i-3}, W_{i-4}))$$

$$Target\ Sentence \equiv \underset{t}{\operatorname{argmax}} \sum_{t=1}^{n} S^t$$

6.1.1.1 Results

We trained this Complete-vocab-Simple N-gram Model, to predict the missing word and its location, on a randomly chosen 3GB data out of total 4GB of training dataset, and tested the model against the remaining 1 GB of the training dataset. Randomly words are popped from the 1GB dataset, so as to generate the validation set. The word location predictor, is able to predict correctly at 22.5% of times. Kaggle ranks the submission on the basis of Levenshtein distance. Such an accuracy would result in poor scores for the challenge. The major drawback of this approach is that it is a mere brute force approach, with no amount of intelligence involved. Such a model takes a large amount to train and test, thus not an ideal model for large corpora. We propose a modified N-gram approach, that restricts the search space of the possible missing word.

6.1.2 Modified N-gram Approach: Supervised Model

N-gram models, have known to capture the domain and the syntactic knowledge about a sentence to a reasonable estimate. N-gram models, thus have been used extensively for language modeling. We also use n-grams as features, but as a modification in a supervised setting.
A supervised approach requires the formulation of the problem as a classification problem, and generally learns a discriminative classifier. The task of finding the location of the missing word here is modeled as a classification problem. Each sentence is processed to capture required features to each possible missing position. The task is thus to classify whether in a test sentence a word is missing or not at that position, given pre-information of the sentence structures, which is gained from the training data.

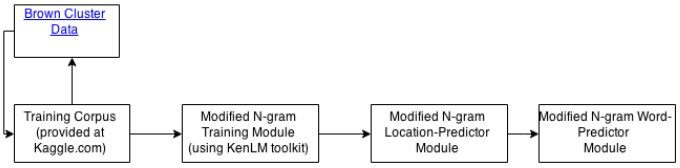

Figure 6.1: Modified N-gram Approach

Relevant features from the training data are extracted by a modified N-gram approach in a supervised manner. This trained knowledge of the corpus is then used to predict the location of the missing word. Once the location of the missing word is predicted correctly, the complexity of problem is reduced many folds. An inherent problem with N-gram models is that they work accurately if the training sentences and test sentences are syntactically similar. But such a model fails in circumstances where the two sets are syntactically different. For example, a training set may capture a trigram like "tomorrow is Wednesday", but it may fail to capture "tomorrow is Tuesday", whose probability to exist in the test set is very high. To capture such semantic similarities, we use a class based approach, wherein words that belong to the same class, possess similar features. In this case days of the week will lie in the the same class. Therefore many semantically similar relations will also be captured by the model, given that one of the class member combination is seen in the training dataset.
This class based grouping of words is achieved through Brown cluster representation for trigram rather than the conventional representation of words in the vocabulary. In this case, days of the week will have similar brown clusters. And those trigram representations, "tomorrow is Wednesday" and "tomorrow is Tuesday", will be very similar. Brown cluster representation for trigram is used rather than representing them using words in the vocabulary. In this case since 'cat' and 'rat' will have similar Brown cluster representation, the trigram 'cat sat on' and 'rat sat on' will be same. Brown clustering would in tern help in better predictions over the unseen data.

6.1.2.1 Model-Preprocessing

This approach of missing word locator is inspired from B. Porwal's work [1] on missing word locator. Training the language model in a supervised setting requires to pre-process the training dataset, and to generate a labeled training dataset.
Randomly a missed word location is picked, and the word at that position is removed from that sentence. N-gram features corresponding to this chosen location are considered to be weighted positive, i.e. by a factor of 1, while all other features are weighted negative, i.e. by a factor of -1. For example consider the sentence "He ran after the black dog". Let the chosen location corresponds to the word "after". The sentence obtained after removing is "He ran the black dog"therefore the word "after" is removed from the sentence. For each sentence, we extract 1 positive feature and (n-1) negative features, where n is the number of words in that sentence.All possible trigram wherein this chosen word can occur are considered as positive elements in the model.
Consider the a trigram model, and removal of the word at position 3 in the model, we calculate 3 possibilities of the missing word corresponding to this position as follows:

- LEFT : 'He ran * the' $\implies$ 'He ran the', +ve feature
- MIDDLE : 'ran * after' $\implies$ 'ran after', +ve feature
- RIGHT : 'ran * the black' $\implies$ 'ran the black', +ve feature

For each possible location as a missing word, we extract a tuples of left, middle and right features, labeling each of them as positive or negative. A feature is labeled positive if they have a missing word and the

[1] Bobby Porwal's missing word locator `https://github.com/nporwal/missing_word_locator`

feature is labeled negative, if they do not have the missing word. Thus each location in the sentence has a corresponding three feature tuple, which are labeled trigram.
A positive label is indication that the trigram is more likely to contain a missing word.Thus for each location, we extract the left, mid and the right features and label them as positive or negative depending on whether they have a missing word. Thus each location consists of three type of features which are trigram and labeled if they are likely to contain the missing word or not. For example for the same example, if we consider for the position 4, the three feature tuple will be as follows:

- LEFT : 'the black dog', -ve feature
- MIDDLE : 'back dog', -ve feature
- RIGHT : 'black dog', -ve feature

where the word-names are a notation wrapper for the brown cluster en-coding obtained earlier. Three feature tuple are collected corresponding to each possible location in a sentence and for all the sentences in the corpus to generate the training datasets.

6.1.2.2 Training Module

Each sentence in the corpus is fed into the model-trainer, which iterates over each possible word location in the sentence. If there is a missing word at that location we update the weight of each component of the corresponding feature tuples by 1, otherwise decrease it by 1.

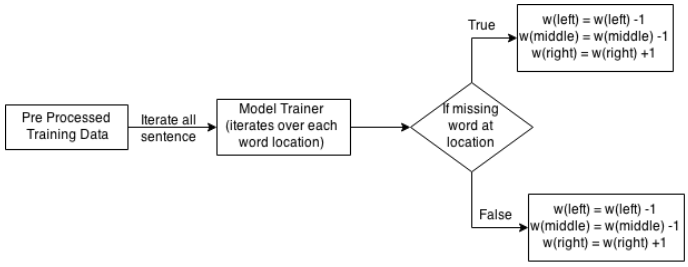

Figure 6.2: Modified N-gram training Module

An inherent drawback of this model is that it never converges and the values are dependent on the distribution of words across the dataset. The average value of the learnt weights over different iterations are taken as the actual weight, giving some sort of stability to the values.

6.1.2.3 Location Predictor Module

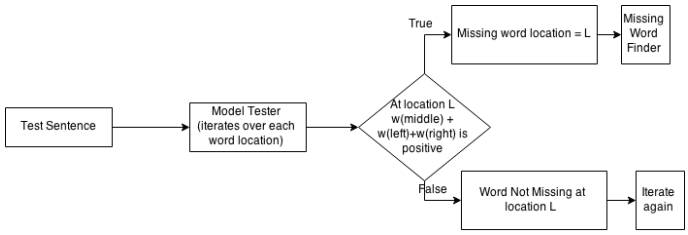

Figure 6.3: Modified N-gram Word-Locator Module

The test dataset consists of new sentences with a word missing at one of its possible locations. The missing word is always from within the sentence, and never at the starting or end of the sentence in consideration.

We apply the Backoff method (Katz et al. '87) [7] and resort to lower order n-grams if the current n-gram is not found in the trained n-gram model. All possible location from the sentence are considered as a viable missing location. If weight count sum of the tuples feature at a particular location, say l is positive, we consider that the position 'l' is the location of the missing word.

6.1.2.4 Results: Word Location Predictor

We trained our Modified N-gram method to predict the missing word location, on a randomly chosen 3GB out of total 4GB of training dataset, and tested the model against the remaining 1 GB of the training dataset. Randomly words are popped from the 1GB dataset, so as to generate the validation set. The word location predictor, is able to predict the location of the missing word correctly 44.5% of times

6.1.2.5 Word Predictor Module

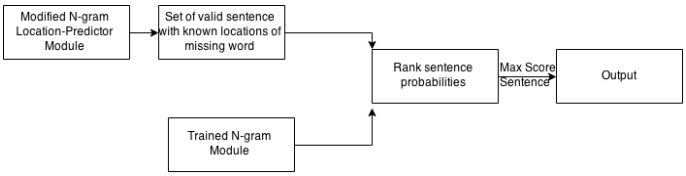

Figure 6.4: Modified N-gram Word-Predictor Module

The N-gram modeled developed without any modifications to the training corpora is used to score the possibility of each of the sentence generated. The highest ranking sentence is chosen as the candidate solution. One of the correct sample outputs generated by the model is as follows

- Actual Sentence: there ***was*** no immediate word of additional damage
- Predicted Sentence : there ***was*** no immediate word of additional damage

Prediction of more frequent words like the determiners and the prepositions are at time predicted incorrectly.

- Actual Sentence: The FDIC established ***the*** TLGP last October
- Predicted Sentence : The FDIC established ***in*** TLGP last October

With nearly half probability the model is able to figure out the missing word location correctly. [2]

6.1.3 MIKOLOV'S WORD2VEC MODEL

The Word2Vec model by Mikolov et al. [2] of late, has become a very popular language model. It converts the words into vectors. Such a vector representation of words could possibly be used to derive several relations between words, and is a starting point of many discoveries in Natural Language Processing. Mikolov et al. [2] proposes two different models for learning word-vector embedding.

- Continuous Bag of Words Model
- Skip Gram Model

[2] More examples will be shortly added, given the computational complexity of the model could not be done at the time of writing the report

The CBOW method is similar to the feed forward NNLM (Neural Network Language model) where the non-linear hidden layer is removed and the projection layer is shared for all words. The model trains the word vector embedding by taking the context words as input to the input layer and the target word as the output. Thus, a one-context CBOW model is like a bi-gram model.

While the skip-gram model introduced by Mikolov et al. [4] is just the reverse of the CBOW model. The target word is input at the input layer and the the context words are at the output layer.

6.1.3.1 CBOW Model

The following understanding of the CBOW model is based from the Xin Rong's explanation of word2vec parameters [6]

As depicted in the figure 6.6, V is the vocabulary size, and the hidden layer size is N (chosen as 100). The nodes on adjacent layers are fully connected. The input word is an one-hot encoding of the word, i.e. for the word x_i in the vocabulary only the i^{th} term in the vector $x_1, x2, ...x_v$ will be 1, and all other would be 0. The weights between the input and the output layer are represented by a VxN matrix W. Each row of W is the N-dimensional vector representation of v_w of the associated word. When computing the hidden layer output, the CBOW model takes the average of the vectors of the input context words, and uses the product input $\Longrightarrow$ hidden weight matrix and the d vector as the output, given mathematically as, where v_{w_I} is the vector representation of the input word w_I:

$$h = \frac{1}{c} * W * (x_1 + x_2 + ... + x_C]$$

$$h = \frac{1}{c} * (v_{w_1} + v_{w_2} + ... + v_{w_C}]$$

where C is the number of words in the context and the words $w_1, ...w_C$ are the words in the context and v_w is the input vector of a word w.

For each word in the vocabulary, there exists an input vector v_w and the output vector $v^{'}_{w}$. Learning the output vector is the process that takes the majority of time of the algorithm. This because in order to update $v^{'}_{w}$, all the words in the vocabulary needs to be iterated upon, which is a very expensive task. Hierarchical soft-max and sampling are used to scale up the word2vec models for larger corpora. Both of these optimize the update computation of output vectors.

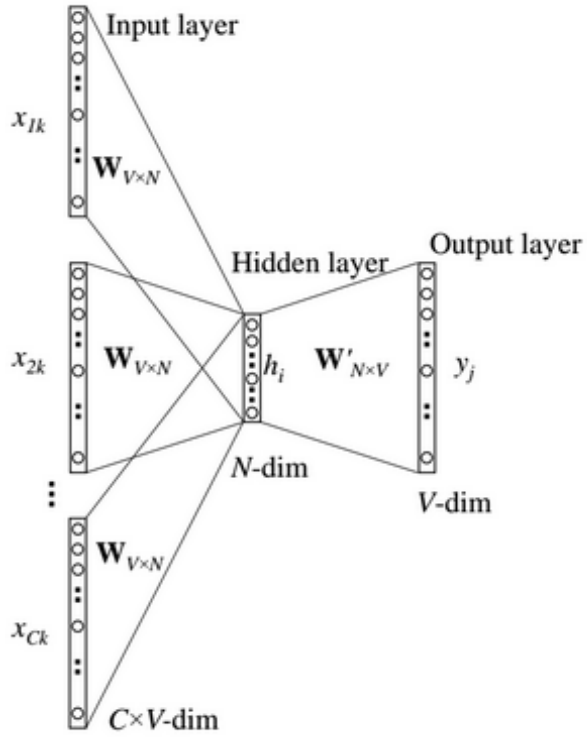

Figure 6.5: Continuous Bag of Words model. Image source link

6.1.3.2 Modified CBOW approach

The following ideas were developed after suggestions and inputs from Mikolov [3], Shashwat Chandra [4], and Pranjal Singh [5]. The training phase of the CBOW model development is very closely related to the objective of this problem. The model trains the word-vector embeddings of a word, through the word-vector embeddings of its neighboring words (context word). The gensim's word2vec model's implementation in python can not be directly used to solve the challenge. The gensim's word2vec code was needed to be suitably modified to model it to be used in favor of the billion word imputation challenge.

[3] Tomas Mikolov Research scientist, Facebook
[4] Shashwat Chandra, M.Tech student, Department of Computer Science and Engineering, IIT Kanpur, India
[5] Pranjal Singh, M.Tech student, Department of Computer Science and Engineering, IIT Kanpur, India

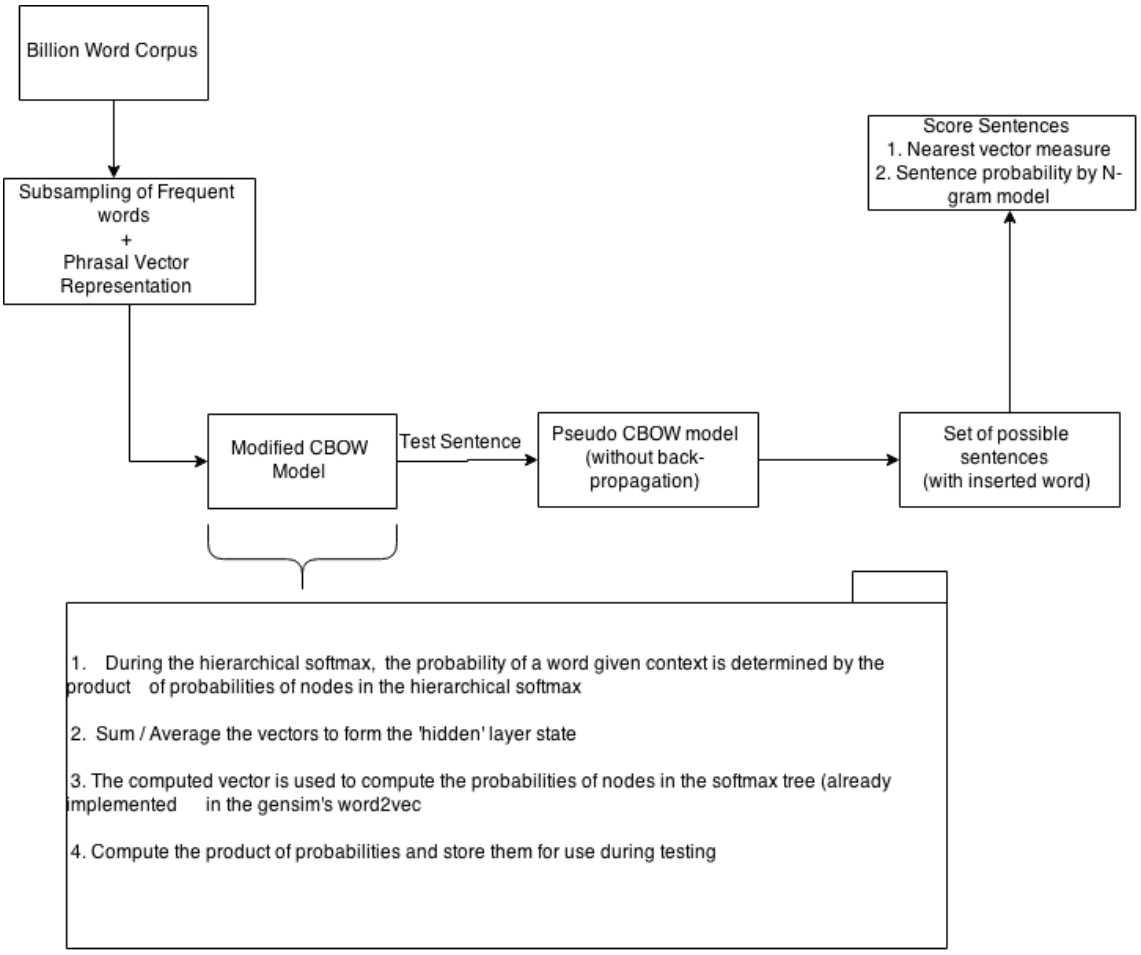

Figure 6.6: Modified CBOW Model

Sub-Sampling of frequent words and bi-gram phrasal vector representation of the words in the vocabulary is carried out to reduce the computational complexity of the model. The gensim word2vec CBOW model with hierarchical softmax is modified for the purpose of this challenge. In the original implementation of the algorithm, the probability of a word given a context, is given by the sum of the words around the word that we are trying to predict, and is determined by the product of probabilities of nodes in the hierarchical softmax (which is represented by a binary tree in word2vec)

After the CBOW model is trained, a pseudo CBOW model is initiated which score a test file, which provides the probability of every word in the test file given its context. Like the original algorithm, the vectors are averaged to form the 'hidden' layer state, which in turn is used to compute the probabilities of the nodes in the softmax tree. Unlike the original algorithm, the product of the probabilities is computed at each stage, (this is not needed for training, but for prediction of words). These probabilities are then stored in a test file, for use during validation.

6.1.3.3 CBOW Model: Testing

Once the CBOW Model is trained accordingly, the saved test file, is used to re-duplicate the CBOW model. Each possible location of missing word in the test sentence is thought of as an output to the trained neural net, and the other as an input context. The probabilities trained from the hierarchical softmax are used to

predict the output word. This prediction may not necessarily lie in the trained word-embedding space. We predict the nearest neighbor of this in the embedding as a possible candidate for missing word. Similarly, candidate solutions corresponding to every possible places of missing words are calculated. These different sentences are then ranked on the basis of the earlier 5-gram model developed. The one scoring the maximum is returned as the candidate solution.

7 Limitations and Challenges

The major challenge and difficulty with this problem is that the task to predict the correct word and the missing location combined is a very difficult task. Multiple words, such as synonyms, antonyms can at times be replaced at the same place. The task to find such a replacement is not possible for a given sentence. Moreover, the greater task still remains to predict both the location and the word simultaneously in meaningful time. Training of a 5-gram model on a large corpus is very time consuming, and also the training is corpora specific. It only captures the syntactic and domain knowledge in the training corpora. Since the models discussed above are highly domain specific, this technique can't be used in run-time environments.

To develop a more general language model, it requires an even greater corpora, whose handling in itself is very time consuming. Strongly supported by the famous Zipf's law [8], it is observed that N-grams become rarer with the increase in the size of corpus. Moreover certain possibilities of sentence structure can't be captured using N-gram models. For example, Ram goes to school on Mondays, and Ram goes to school on Tuesdays, such type of relations can't be easily captured.

8 Possible Approaches

The challenge is very computing intensive, For example the Cantab's approach as mentioned on Kaggle-Forum, involves a training time of couple of weeks. Such a large training time, makes it a really time-inefficient approach. A possible approach could be the use of Conditional Random Fields to incorporate structural information. Parse trees and Part of Speech tagging can be used to limit the search space of possible words. Recurrent neural networks are famous for language modeling tasks, but the only drawback they suffer is from large training time, spanning about 10 days. Once the model is trained on the corpus, parse trees and part of speech information can be used to limit the search space of the possible missing words.

References

[1] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, Tony Robinson, One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling `http://arxiv.org/abs/1312.3005`

[2] Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013)

[3] Will Williams, Niranjani Prasad, David Mrva, Tom Ash, Tony Robinson, Scaling Recurrent Neural Network Language Models, `http://arxiv.org/abs/1502.00512`

[4] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[5] `https://github.com/percyliang/brown-cluster`

[6] Xin Rong, word2vec Parameter Learning Explained `https://github.com/percyliang/brown-cluster`

[7] S.M. Katz, Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer,in IEEE Transactions on Acoustics, Speech and Signal Processing, volume ASSP-35,pages 400-401,March 1987

[8] Christopher D. Manning, Hinrich SchÃijtze Foundations of Statistical Natural Language Processing, MIT Press (1999), ISBN 978-0-262-13360-9, p. 24

[9] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in Neural Information Processing Systems. 2013

[10] Microsoft Sentence Completion Challenge `http://research.microsoft.com/apps/pubs/default.aspx?id=15703`

[11] Mnih,Andriy, and Koray Kavukcuoglu. "Learning word embeddings efficiently with noise-contrastive estimation." Advances in Neural Information Processing Systems. 2013