Context extraction for in service level agreements using skip gram and neural network

Divya Natolana Ganapathy

1 Introduction

The National Institute of Standards and Technology (NIST) has been designated by the Federal Chief Information Officer (CIO) to accelerate the federal government's secure adoption of cloud computing by leading efforts to identify existing standards and guidelines. Among these standards are the standards for Service Level Agreements(SLA) . In this project we focus on extracting one section of the SLA which is definitions. The cloud providers must define the key terms that would be used in the agreement called definitions. However, different cloud providers use different words in their agreement. For Example, AWS uses "Credit Request" , Microsoft Azure uses "Claims", for the rebate that would be paid back to the customer in situation of service unavailability. AWS uses "Unavailability", Google Cloud Provider uses "Downtime" for the period of service not being available. Now, extracting different terms of similar meaning from different documents would have been trivial if there was trained knowledge available , like for example synonyms in regular English words. However, extracting the above mentioned words from a cloud domain SLA becomes difficult as unavailability and downtime mean the same only in cloud domain. This context of words can be extracted using skip gram which is built with the help of a neural network. As the name suggested , skip gram involves skipping the word we are interested in and picking its neighbours to the right and left in the document. These neighbours form the label to the input word. The neural network weights will give the word embedding for these words and the output layer would be used to get the probability of a word being a context work for the given input center word.

2 Related Work

Context extraction or extracting the semantic similarity of words has been a key are of research from when NLP began. There have been several attempts and procedures for identifying words that mean the same semantically. One such approach is shown in Speech and Language Processing. Daniel Jurafsky James H. Martin. Which specifically talk about using bi gram for extracting the context of a word. It divided the bi-gram into key phrases and themes. The themes specified the context of words e.g. If you stop "cold stone creamery", the phrase "cold as a fish" will make it through and be decomposed into n-grams as appropriate, if we instead use 'cold stone ' bigram then the second phrase will not go through. Later in the work Not All Contexts Are Created Equal: Better Word Representations with Variable Attention by Wang Ling Lin Chu-Cheng Yulia Tsvetkov Silvio Amir Ramon Fernandez Astudillo Chris Dyer Alan W Black Isabel Trancoso Continuous Bag of Word predicts the probability of a word given a context. A context may be a single word, or a group of words, the activation function is like what is used in this experiment, one disadvantage of this is that CBOW takes the average of the context of a word (as seen above in calculation of hidden activation). For example, Apple can be both a fruit and a company but CBOW takes an average of both the contexts and places it in between a cluster for fruits and companies. This method also takes a very long time to train.

3 Methodology

3.1 Data Collection

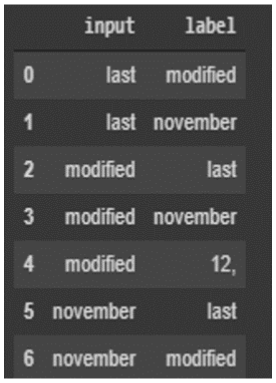
The data we use in this experiment is the cloud compute service level agreement (SLA) of various cloud service providers like AWS, Azure, VMware, Kamatera, Google cloud Platform etc. This is collected by web scraping various cloud service provider's publicly available SLA using NLP library 'Beautiful Soup'. This collected data from various websites is then aggregated to form a single text corpus which is around 50 KB of data.

3.2 Primary Data Preprocessing

The corpus obtained needs to be cleaned and made into a suitable training dataset. From the application we know that we do not want any stopwords like 'and', 'the', 'is', 'on' etc. The presence of these words acts as noise and hamper the distribution of other important words like 'uptime', 'availability' etc. hence, the stopwords are discarded. We get rid of punctuations, newline and tab characters and words that are longer than 14 letters or lesser than 2 letters, as these do not contribute positively to the result. These words are then iterated through each sentence and tokenize the words using nltk word tokenizer. The data is split half into training and half into test dat. 1/3rd of training data is considered validation data.

3.3 Secondary Data Preprocessing- Preparing the data for skip gram and neural network

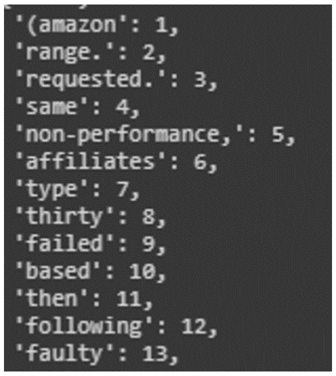
Input and label pairs are created by taking each word as the input word and the neighbors of the word as labels of this word. The number of labels depend on the window size, if window size is 1 then we have 2 labels per word on either side of the word and so on. For e.g. the sentence "Last modified November 12" would be (Input : Last) (Label: modified) , (Input : Last) (Label: November) as shown in 1



```
         input     label
0         last  modified
1         last  november
2     modified      last
3     modified  november
4     modified       12,
5     november      last
6     november  modified
```

Fig. 1. Input and Label

Using this we build the co-occurrence vectors. Each word needs to be converted into numbers or vectors, using which we can infer how far or close a word is from another word. Closer words are more in context of a word compared to farther words. An array is created using the word as the first element and its position in the text as the second element as shown in 2



```
'(amazon': 1,
'range.': 2,
'requested.': 3,
'same': 4,
'non-performance,': 5,
'affiliates': 6,
'type': 7,
'thirty': 8,
'failed': 9,
'based': 10,
'then': 11,
'following': 12,
'faulty': 13,
```

Fig. 2. words with position values

This way every word gets a number , even same words appear second or third time they will have different number. Once this is done, we create the one hot encoding of every word as the input to the neural network should be numbers and not text. In one hot encoding , an array of zeros is created, except for the element in the array which holds the position of the word in the document. This way every word is represented by an array of the size equal to the length of the document. We take a set of random words as test data and the rest as training data. One third of the training data is treated a validation data. The window size chosen here is 2. It could be varied to get the best possible prediction.

3.4 Training the Neural network

3.4.1 Forward Propagation

A three-layer neural network is built with first layer as the input layer, the input to this is one hot encoded word. The input to the first layer could look like [0 0 0 0 1 0 ..] The second layer is the hidden layer. Here the input layer neuron values are multiplied by a matrix of weight vectors. [0 0 0 0 1 0 ..] (x) * [weight matrix] = [some value] say h

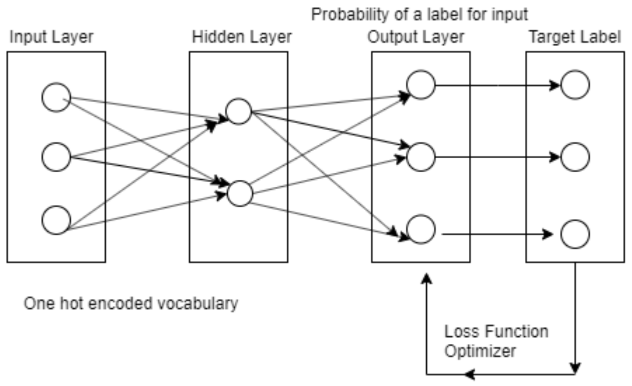
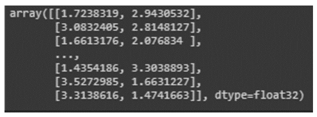


Fig. 3. Neural network model

Which is summed over all the input values to give the final second layer neuron value. In the beginning it is started by randomized weights, these weights keep getting updated by the optimizer until a desired output is obtained. These weights are the actual embedding of the words. The activation of this layer is the input along with next set of weights for the third layer as shown in 4

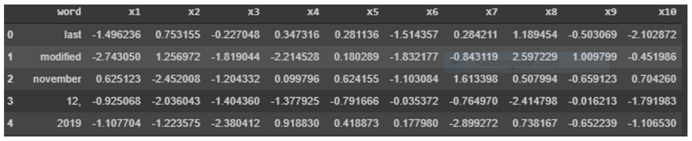
$$h = W_{input}^{T} \cdot x \in \mathbb{R}^{N}$$

The third layer is the output layer , which with the help of a non-linear function softmax produces the probability of a set of words being the neighbor of the input word. The output will represent the words as a matrix of weights, the dimension of this matrix can be varied in the program. In the current experiment dimension 10 as well as a dimension of 2 is taken. If the dimension is 2 then every word is represented by a 2-dimensional vector and can be plotted on a 2D graph. Whereas if dimension 10 is taken then each word is represented by a 10-dimensional matrix number. However, plotting this output matrix is hard on a 2D scale therefore PCA and t-NSE is used to reduce the dimension to a 2D plot.the 10 and 2 dimensional embedding are as shown in 5 and 6



```
array([[1.7238319, 2.9430532],
       [3.0832405, 2.8148127],
       [1.6613176, 2.076834 ],
       ...,
       [1.4354186, 3.3038893],
       [3.5272985, 1.6631227],
       [3.3138616, 1.4741663]], dtype=float32)
```

Fig. 5. 2-D embeddings



	word	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
0	last	-1.496236	0.753155	-0.227048	0.347316	0.281136	-1.514357	0.284211	1.189454	-0.503069	-2.102872
1	modified	-2.743050	1.256972	-1.819044	-2.214528	0.180289	-1.832177	-0.843119	2.597229	1.009799	-0.451986
2	november	0.625123	-2.452008	-1.204332	0.099796	0.624155	-1.103084	1.613398	0.507994	-0.659123	0.704260
3	12,	-0.925068	-2.036043	-1.404360	-1.377925	-0.791666	-0.035372	-0.764970	-2.414798	-0.016213	-1.791983
4	2019	-1.107704	-1.223575	-2.380412	0.918830	0.418873	0.177980	-2.899272	0.738167	-0.652239	-1.106530

Fig. 6. 10-D embeddings

3.4.2 Output Activation function

The output layer is a V dimensional probability distribution of the words given a center word. The probability of A given B P (A—B) , P (w context — w center) can be represented as shown in 7. This is obtained by softmax function . The exponential term in the numerator makes all the values positive , the normalization factor in the denominator makes sure the value is in between [0 1] as shown in 8

$$p(w_{context}|w_{center}) = \frac{exp(W_{output_{(context)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)} \in \mathbb{R}^1$$

Fig. 7. Softmax Function

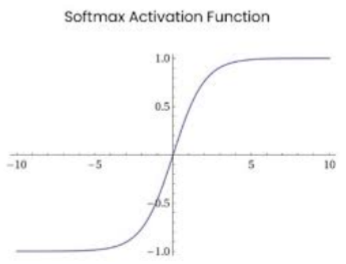


Fig. 8. Softmax Curve

The output is thus, the conditional probability of finding words given the center word. We maximize the probability , P(w context — w center) if favorable using 9

$$\underset{\theta}{\text{argmax}}\ log \prod_{c=1}^{C} \frac{exp(W_{output_{(c)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)}$$

Fig. 9. Maximize the softmax function

3.4.3 Cost function

Cross entropy(log loss)- This measures the performance of classification whose output is in between 0 and 1. The loss increases when the predicted probability is far away from the actual label and vice versa. We minimize the cost function This is calculated using the formula as shown in 10 The curve and graph for cross entropy is as shown in 11

3.4.4 Backward Propagation

Backpropagation is used by neural network to signal the error in the backward direction from output to input. This is done to alter the parameters to reduce the error between the expected and the predicted value. Here our parameter is the weight vector. We are altering the word embeddings to

$$J(\theta; w^{(t)}) = -log \prod_{c=1}^{C} \frac{exp(W_{output_{(c)}} \cdot h)}{\sum_{i=1}^{V} exp(W_{output_{(i)}} \cdot h)}$$

Fig. 10. Minimize the cost function

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Fig. 11. expression for cross entropy

closely match to what we expect in the result. In this current experiment gradient descent is used in back propagation. The weight updating using gradient descent is done using 12

$$\theta_{new} = \theta_{old} - \eta \cdot \nabla_{J(\theta; w^{(t)})}$$

Fig. 12. Weight updating using Gradient Descent

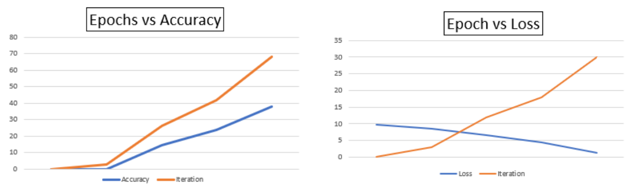


Fig. 13. Effect of epochs on loss and accuracy

The step size is the learning rate, after a few trail and error we found that the best results were obtained using 0.05 as the step size. The value to be updated is obtained by the partial derivative of the loss function. new is the updated weight, old is the older weight. This is done until the error reduces over multiple iterations. The effect of epochs on the loss and accuracy is shown below in 13. The loss is reducing, and accuracy is increasing with increase in the number of iterations.

3.5 Measurements and Results compared to a baseline

A better measurement of a system like this is visualize the position of words. When we choose the dimension of weight vector if we choose 2 dimensions then it can be plotted on a simple scatter plot directly. However, if larger dimensions like 10 or even 100 in case of genism's word2Vec we need to reduce the dimensionality of the result. This will help us project it onto a 2 D graph. Both the scenarios are visualized and shown in 16 Accuracy obtained is around 40 percentage however, when visualized we see that words of a context from different documents lie close to each other. Dimensionality reduction – t-nse , t- Distributed Neighbor Stochastic Embedding is a machine learning visualization technique used to visualize very high dimensional data into 2 or 3 dimensions. It reduces the dimensionality in such a way that similar objects are modeled by nearby points and dissimilar objects by farther points. The word embeddings

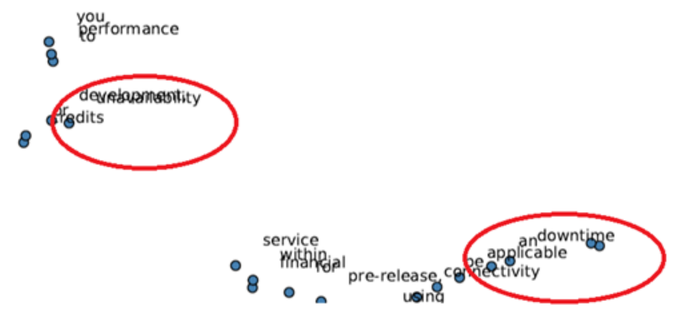
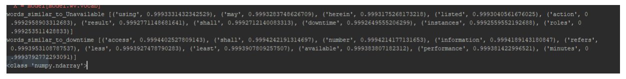


Fig. 14. Current experiment visualization using t-SNE. 'Unavailability' and 'downtime' are in close proximity



```
 X = model[model.wv.vocab]
words_similar_to_Unavailable [('using', 0.9993331432342529), ('may', 0.9993283748626709), ('herein', 0.9993175268173218), ('listed', 0.9993040561676025), ('action', 0
 .9992958903312683), ('result', 0.9992771148681641), ('shall', 0.9992712140083313), ('downtime', 0.9992649555206299), ('instances', 0.9992559552192688), ('roles', 0
 .999253511428833)]
words_similar_to_downtime [('access', 0.9994402527809143), ('shall', 0.9994242191314697), ('number', 0.9994214177131653), ('information', 0.9994189143180847), ('refers',
 0.9993953108787537), ('less', 0.9993927478790283), ('least', 0.9993907809257507), ('available', 0.999383807182312), ('performance', 0.999381422996521), ('minutes', 0
 .9993792772293091)]
<class 'numpy.ndarray'>
```

Fig. 15. words most similar to 'unavailability' and 'downtime'

obtained have closer values for in context words, the Euclidean distance between in context words are lesser than that of words that do not belong to a context.

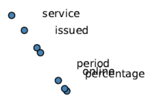


Fig. 16. word 'percentage' is close to words like 'service' , 'period' , these words are used in the context of calculating the percentage of time a service is up

As seen 16 the words 'unavailability' and 'downtime' are from 2 different documents AWS SLA and GCP Sla respectively, but the word embeddings fall very close to each other, their word embedding values are closer to each other making them fall very close to each other in the final weights distribution of the hidden layer. This shows that we have successfully captured the context of these words using skip gram and neural networks. This could be applied to any new word used in future to refer to 'unavailability' . this way we have built a system which helps in the machine to understand documents of this domain even with new unknown words. The results are compared with that obtained from word2Vec implementation of genism, we observe that the final word embedding is not able to perform as good as the current experiment for the SLA data set. This could be because the wor2Vec implementation only lets us change the window size or iterations and does not let us alter the parameters or choose the loss function or even the back propagation. Word2Vec is a complete black box and we not know what exactly is happening inside, whereas by building our own skip gram and feeding it to neural network has led us after trial and error in picking the right activation function, cost function and even the backpropagation and the step size in back propagation.

4 Conclusion and Future Work

In this experiment semantically similar words have been successfully grouped closer to each other using prediction-based context extraction. Each word from each document has been captured along with its neighbors. This creates an input label pair which could then be fed to the neural net using one hot encoding method. In forward propagation the weighted sum of neurons from the previous layer form the value of the current neurons. At the output layer the probability of words in the corpus being the context word of the input word is obtained. Softmax function is used as the output layer activation function. The cost function is calculated using cross entropy and is optimized using gradient descent in back propagation. The updating of weights using gradient descent is done in a step size of 0.05 until desirable results are obtained (sufficient loss reduces). The weights in the hidden layer gives the final embedding of the words. This when in high dimension, plotted using a dimensionality reduction method (PCA , t-NSE) gives the visual representation of the distribution. This experiment has successfully identified and grouped closely related (in context) words together. The model gives an accuracy of 40 percent but clearly identifies words like unavailability and downtime from different documents as closely related words. In future, this work could be extended to improve the accuracy using better data set or bigger data set and incorporate different domains.

5 References

- Not All Contexts Are Created Equal: Better Word Representations with Variable Attention - https://www.cs.cmu.edu/ ytsvetko/papers/emnlp15-attention.pdf
- Distributed Representations of Words anD Phrases and their Compositionality https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
- https://aegis4048.github.io/demystifyiNg_neural_network_in_ski
- https://github.com/minsukheo/python_tutorial/blob/master/da
- Speech and Language Processing. Daniel Jurafsky James H. Martin https://web.stanford.edu/ jurafsky/slp3/3.pdf
- https://towardsdatascience.com/how-i-used-natural-language-processing-to-extract-context-from-news-headlines-df2cf5181ca6
- https://aws.amazon.com/compute/sla/
- https://cloud.google.com/compute/sla
- https://www.kamatera.com/Service_Level_Agreement
- https://cloud.ibm.com/docs/overview?topic=overview-slas
- http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
- http://www.oracle.com/us/corporate/contracts/saas-online-csa-us-1894130.pdf
- https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
- word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method https://arxiv.org/abs/1402.3722
- Enriching Word Vectors with Subword Information https://paperswithcode.com/paper/enriching-word-vectors-with-subword