Word Embeddings - Skip Gram Model

P. Preethi Krishna(✉) and A. Sharada

CSE Department, G. Narayanamma Institute of Technology and Science, Hyderabad, Telangana, India
krishna.preethi95@gmail.com, sharada@gnits.ac.in

Abstract. Word embedding is of great importance for any NLP task. Word embeddings is used to map a word using a dictionary to a vector. Skip gram model is a type of model to learn word embeddings. This model will predict the surrounding words based on the given input words which are within the given distance. It aims to predict the context from the given word. Words occurring in similar contexts tend to have similar meaning. Therefore it can capture the semantic relationship between the words. This paper explains about the word embedding using skip gram model, its architecture and implementation.

Keywords: Word embedding · Skip gram model · Hot encoded vector

1 Introduction

The web has a voluminous vocabulary of words. Each word gives a subjective and objective meaning for a sentence. Every word can be sensed differently based on the situation or context [11]. With the rapid inclusion of Natural Language Processing (NLP) tasks [2] there is a need to consider all words, relationships between words, synonyms, and antonyms based on context. Instead of machine learning methodologies deep learning methodologies are being considered in all research works. Deep learning considers the neural network structure which consists of neurons as our basic element to work on [6]. At a stretch we can work on a huge amount of data by using these neurons. So for embeddings of large words this skip gram model is a good choice [5].

1.1 Word Embeddings

NLP requires words to be represented in numerical format to do manipulations as it is incapable to process any text or string [7]. Word embedding will map a word to a vector using a dictionary [13]. The meaning of a word can be approximated by the set of contexts in which it occurs. Words with similar vectors are semantically similar in meaning as vector encoding capture the semantic of the word. The vector representation is termed as *hot encoded vector*. Hot encoded vector is represented with only 0's and 1's. 1 is represented for the position of the input word in the sentence and 0 for all other words in that sentence. These vectors help us to encode the semantic relationship among the other words.

V. K. Gunjan et al. (Eds.): *ICICCT 2019 – System Reliability, Quality Control, Safety, Maintenance and Management*, pp. 133–139, 2020.
https://doi.org/10.1007/978-981-13-8461-5_15

Example representation for hot encoded vector. For the sentence "The cat jumped over the puddle",

The = [1 0 0 0 0 0]
Cat = [0 1 0 0 0 0]
Jumped = [0 0 1 0 0 0]
Over = [0 0 0 1 0 0]
The = [0 0 0 0 1 0]
Puddle = [0 0 0 0 0 1]

Word embeddings can perform few tasks like finding the degree of similarity between two words, finding odd one out, probability to find a text under the document etc. Few applications of word embedding are like machine translation, sentiment analysis, named entity recognition, chat bots and so on.

1.2 Skip Gram Model

Skip gram model aims to predict the context words for given input word. The idea behind skip gram model is to take a word and predict all the related contextual words. In other words, it will predict the context when a word is given. Skip gram model is built on word embeddings. It's an extended version of N-gram model where instead of including consecutive words we can go for skipping of words while considering mapping.

In skip gram model a simple neural network with a single hidden layer is used. Main intuition behind this model is that given a word $w^{<k>}$ at the k^{th} position within a sentence it will predict the most probable surrounding context. The word is represented as its index 'i' within the vocabulary V and fed into a projection layer that turns this index into a continuous vector given by the corresponding i^{th} row in the layer weight matrix.

Skip gram model belong to prediction-based vector which works more efficiently with small training dataset. Even infrequent words are also well presented using this model. Words occurring in similar contexts tend to have similar meanings. Therefore, it can capture the semantic relationship between the words. So, this model is like a simple logistic regression (Softmax) model.

2 Architecture

Skip gram model architecture considers a vocabulary of all distinct words [1, 7, 8, 14, 15]. These distinct words are fed into the input layer of the model. The number of nodes in the hidden layer represents the dimensionality of the system. Hidden layer is represented as a weight matrix. In weight matrix rows count is the number of words present in the vocabulary and column count is the number of neurons.

For example, if we are learning with 300 features then will have 300 columns. If we have 10000 rows one for each word in vocabulary then the weight matrix will be like:

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

The evaluation in hidden layer is just like a lookup table. The output layer is a Softmax regression classifier. Each output neuron i.e. one per word in vocabulary will produce an output between 0 and 1 and the sum of all these output values will sum up to 1. So, in skip gram model target word is fed at the input, the hidden layer remains the same and the output layer is replicated multiple times to accommodate the input and context words (Fig. 1) [14].

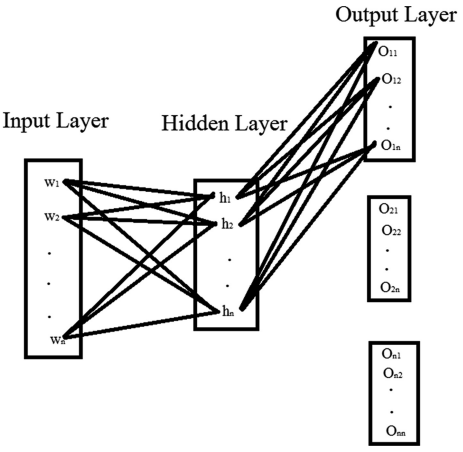

Fig. 1. Skip gram model architecture

3 Working Principle

The training objective of the Skip-Gram model is to learn word vector representations that are good at predicting nearby words in the associated context. This model has an input layer, a single hidden layer and an output layer. Hidden layer is made of neurons. All unique words in vocabulary are given to input layer. We select a central word to perform the mapping. For the selected central word search is performed to find the nearest words in sequence, semantically or logically related words. The input to the network is encoded using "1-out of-V" representation meaning that only one input line is set to one and rest of inputs are set to zero.

Firstly, a vocabulary of words should be created. Then encode it as vectors of 0's & 1's. The functionality is like; each current word i.e. target word is passed to the input layer which is further passed to the hidden layer. Simply the word is copied to the hidden layer. Now based on the skip window size the mapping to related words is done. These are passed to the output layer. Here at output layer the pair of words (target, context) with sum equivalent to one is considered as the output.

3.1 Implementation

Simple steps involved for implementation of skip gram model:

(i) Build a corpus or dataset & vocabulary which should be used. A vocabulary is like a dictionary with all distinct words from corpus should be arranged in alphabetical order. Indexes are assigned for the unique words. Preprocessing step of case normalization, removing space, removing punctuation, tokenization should be done. This vocabulary is helpful like a look-up table for mapping words to meaning.

(ii) Build a skip-gram generator of format (target, context) where target word is the word for which we need to find the neighboring words which are the required context words. In the output when label value is 0 it means it is irrelevant else if value is 1 means it's relevant.

(iii) Build the skip gram model architecture where at input layer skip gram generator format input is passed to get the related context words at the output layer.

(iv) Train the model to get the functionality run even when new words are added. After training similar words with similar weights gives out the same values as similarity words.

Input matrix representation is as [8]:

W_{11}	W_{12}	W_{13}
W_{21}	W_{22}	W_{23}
W_{31}	W_{32}	W_{33}
W_{41}	W_{42}	W_{43}
W_{51}	W_{52}	W_{53}

Where,

W_{11} – weight of neuron from a node w_1 to h_1

W_{12} – weight of neuron from a node w_1 to h_2

W is weight defined in input layer for the word

h is the weight defined in hidden layer for that word

Function of input to hidden layer connection is basically to copy the input word vector to hidden layer. We define a window called "skip-window" which is the number of word movement back and forth from the selected words. The input words are converted to a numerical representation.

The output matrix is represented as:

W'_{11}	W'_{12}	W'_{13}	W'_{14}	W'_{15}
W'_{21}	W'_{22}	W'_{23}	W'_{24}	W'_{25}
W'_{31}	W'_{32}	W'_{33}	W'_{34}	w'_{35}

Where,

W'_{11} – weight of neuron from a node h_1 to O_{11}

h is the weight in hidden layer

o is the weight in output layer

The input matrix and output matrix values are initialized to some small random values as per the neural network training method.

After all the steps are finished and plotted on a graph we can see that all similar words are grouped together, and dissimilar words are plotted apart.

Evaluation/Example

Consider the sentences, “the dog saw a cat”, “the dog chased the cat”, “the cat climbed a tree” [10, 12]. The corpus vocabulary has eight words when ordered alphabetically. Following are the eight words in the vocabulary:

A
Cat
Chased
Climbed
Dog
Saw
The
Tree

The skip gram generator format for the “the dog saw a cat” sentence will be (the, dog), (the, saw), (the, a), (the, cat), (dog, saw), (dog, a), (dog, cat), (saw, a), (saw, cat), (a, cat). Similarly, for other sentences also this format is generated (Fig. 2) [12, 15].

For the sentence “the cat climbed a tree”, the input in the form of (word, target) will be (the, cat), (the, climbed), (the, a), (the, tree), (cat, climbed), (cat, a), (cat, tree), (climbed, a), (climbed, tree), and (a, tree). These are passed to hidden layer in hot coded vector format from input layer. Skip gram model is based on context so when the input is cat and target is climbed it will find all its related and nearest words. When the hidden layer finds the map (cat, climbed) it had successfully searched the target based on context input provided. So input vector is [0 1 0 0 0 0 0 0] and output vector is [0 0 0 1 0 0 0 0].

To improve the accuracy with respect to finding target word based on multiple context words we can include a parameter called “window”. If the value of window is 5 then for the given input word it will find 5 nearest words pointed as target. When a perfect match of word and target is found then it is the required output from the model.

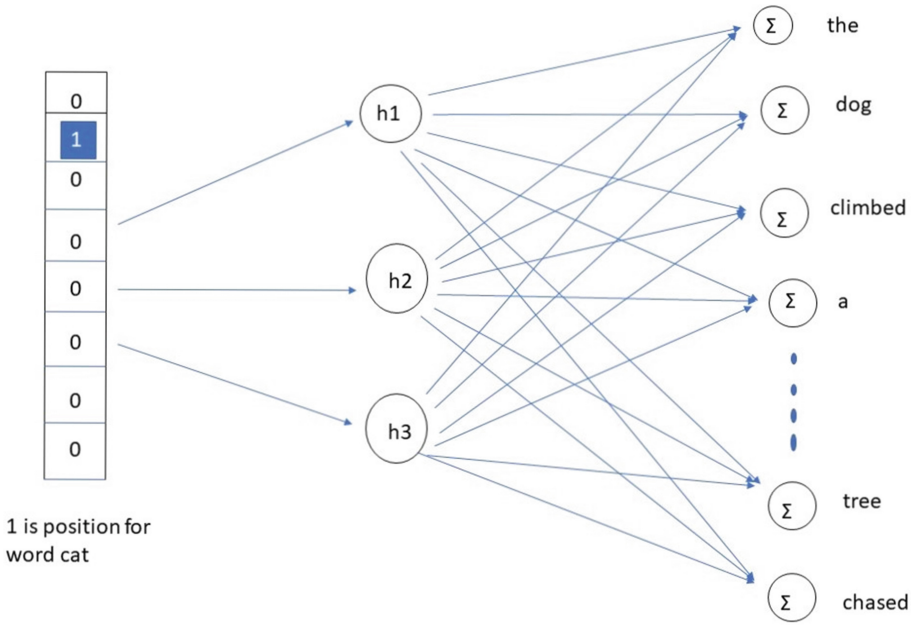

Fig. 2. Representation of example

Output of the k^{th} neuron is computed as

$$Y_k = P_r(word_k|word_{context}) = \exp(activation(k)) \div \sum_{n=1}^{v} \exp(activation(n))$$

Where activation (n) represents the activation value of the n^{th} output layer neuron [4, 9].

4 Conclusion

Skip gram model is useful for word embedding where we can find out surrounding context words for the given input or target word [3]. In other words, it outputs the contextually related words for the given input word. This model is built on concept of neurons because neurons help to do large computation with effective performance and efficiency. It can even capture two semantics for a single word. It can be used for sentiment analysis from multidomain. This model works well with a small amount of the training data, even with rare words or phrases. This word embedding is of much use nowadays for NLP tasks to be carried out. Word embedding is used to figure out better word representations than the existing ones.

References

1. Alex Minnaar - Word2Vec Tutorial part I: The skip-gram model, April 2015
2. Chaubard, F., Mundra, R., Socher, R.: CS 224D: Deep Learning for NLP1 1 Lecture Notes: Part I2 2. Spring (2016)
3. Guthrie, D., Allison, B., Liu, W., Guthrie, L., Wilks, Y.: A closer look at skip-gram modelling. In: LREC (2006)
4. Maillard, J., Clark, S.: Learning adjective meanings with a tensor-based skip-gram model. In: 19th Conference on Computational Language Learning, Beijing, China, 30–31 July, pp. 327–331 (2015)
5. Ma, L., Zhang, Y.: Using Word2Vec to process big text data. In: 2015 IEEE International Conference on Big Data (Big Data) (2015)
6. Dragoni, M., Petrucci, G.: A neural word embeddings approach for multi-domain sentiment analysis. IEEE Trans. Affect. Comput. **8**(4) (2017)
7. Bhoir, S., Ghorpade, T., Mane, V.: Comparative analysis of different word embedding models (2017)
8. Levy, O., Goldberg, Y.: Dependency-based word embeddings
9. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems, vol. 26, October 2013
10. Word embedding. sebastianruder.com/word-embedding-1/. Accessed 15 Aug 2016
11. Goldberg, Y., Levy, O.: Word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method (2014)
12. http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
13. http://www.thushv.com/natural_language_processing/word2vec-part-1-nlp-with-deep-learning-with-tensorflow-skip-gram/
14. https://iksinc.online/tag/skip-gram-model/
15. https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b