|  |  |
| --- | --- |
| Name Of The Student | Aman Rai |
| Internship Project Topic | Automate Detection of different emotions from textual comments and feedback |
| Name of the Organization | TCS iON |
| Name of the Industry Mentor | Mr. Debashis Roy |
| Name of the Institute | Institute of Engineering & Management Kolkata |

|  |  |  |
| --- | --- | --- |
| Date | Day # | Hours Spent |
| 27 August 2020 | 7 | 4+ |
| Activities done during the day: Today I am going to learn about Word2Vector  **Problem with BOW & TF-IDF**   * Both BOW (Bag of Words) and TF-IDF approach semantic information is not stored. TF-IDF gives importance to uncommon words * There is definitely chance of over-fitting   **Solution Word2Vec**   * In this specific model each word is basically represented as a vector of 32 or more dimension instead of a single number. * Here the semantic information and relation between words is also preserved.     **Steps to create Word2Vector**   * Tokenization of the Sentences * Create Histograms * Take most frequent Words * Create a matrix with all unique words. It also represent the occurrence relation between the words.   Word embedding is one of the most popular representation of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc.  What are word embeddings exactly? Loosely speaking, they are vector representations of a particular word. Having said this, what follows is how do we generate them? More importantly, how do they capture the context?  Word2Vec is one of the most popular technique to learn word embeddings using shallow neural network. It was developed by Tomas Mikolov in 2013 at Google.  **Need of Word Embedding**  Consider the following similar sentences: *Have a good day*and *Have a great day.*They hardly have different meaning. If we construct an exhaustive vocabulary (let’s call it V), it would have V = {Have, a, good, great, day}  Now, let us create a one-hot encoded vector for each of these words in V. Length of our one-hot encoded vector would be equal to the size of V (=5). We would have a vector of zeros except for the element at the index representing the corresponding word in the vocabulary. That particular element would be one. The encodings below would explain this better.  Have = [1,0,0,0,0]`; a=[0,1,0,0,0]` ; good=[0,0,1,0,0]` ; great=[0,0,0,1,0]` ; day=[0,0,0,0,1]` (` represents transpose)  If we try to visualize these encodings, we can think of a 5 dimensional space, where each word occupies one of the dimensions and has nothing to do with the rest (no projection along the other dimensions). This means ‘good’ and ‘great’ are as different as ‘day’ and ‘have’, which is not true.  Our objective is to have words with similar context occupy close spatial positions. Mathematically, the cosine of the angle between such vectors should be close to 1, i.e. angle close to 0.    Here comes the idea of generating distributed representations. Intuitively, we introduce some dependence of one word on the other words. The words in context of this word would get a greater share of this dependence. In one hot encoding representations, all the words are independent of each other, as mentioned earlier.  **How Word2Vector Works**  Word2Vec is a method to construct such an embedding. It can be obtained using two methods (both involving Neural Networks): Skip Gram and Common Bag Of Words (CBOW)  **CBOW Model:**This method takes the context of each word as the input and tries to predict the word corresponding to the context.  **Skip Gram:** This looks like multiple-context CBOW model just got flipped. To some extent that is true. We input the target word into the network. The model outputs C probability distributions. What does this mean? For each context position, we get C probability distributions of V probabilities, one for each word. In both the cases, the network uses back-propagation to learn.  Both have their own advantages and disadvantages. According to Mikolov, Skip Gram works well with small amount of data and is found to represent rare words well.  On the other hand, CBOW is faster and has better representations for more frequent words.  The above explanation is a very basic one. It just gives you a high-level idea of what word embeddings are and how Word2Vec works.  Code of Word2Vector  import nltk  from gensim.models import Word2Vec  from nltk.corpus import stopwords  import re  paragraph = """I have three visions for India. In 3000 years of our history, people from all over  the world have come and invaded us, captured our lands, conquered our minds.  From Alexander onwards, the Greeks, the Turks, the Moguls, the Portuguese, the British,  the French, the Dutch, all of them came and looted us, took over what was ours.  Yet we have not done this to any other nation. We have not conquered anyone.  We have not grabbed their land, their culture,  their history and tried to enforce our way of life on them.  Why? Because we respect the freedom of others.That is why my  first vision is that of freedom. I believe that India got its first vision of  this in 1857, when we started the War of Independence. It is this freedom that  we must protect and nurture and build on. If we are not free, no one will respect us.  My second vision for India’s development. For fifty years we have been a developing nation.  It is time we see ourselves as a developed nation. We are among the top 5 nations of the world in terms of GDP. We have a 10 percent growth rate in most areas. Our poverty levels are falling.  Our achievements are being globally recognised today. Yet we lack the self-confidence to  see ourselves as a developed nation, self-reliant and self-assured. Isn’t this incorrect?  I have a third vision. India must stand up to the world. Because I believe that unless India  stands up to the world, no one will respect us. Only strength respects strength. We must be  strong not only as a military power but also as an economic power. Both must go hand-in-hand. My good fortune was to have worked with three great minds. Dr. Vikram Sarabhai of the Dept. of space, Professor Satish Dhawan, who succeeded him and Dr. Brahm Prakash, father of nuclear material. I was lucky to have worked with all three of them closely and consider this the great opportunity of my life. I see four milestones in my career"""  # Preprocessing the data  text = re.sub(r'\[[0-9]\*\]',' ',paragraph)  text = re.sub(r'\s+',' ',text)  text = text.lower()  text = re.sub(r'\d',' ',text)  text = re.sub(r'\s+',' ',text)  # Preparing the dataset  sentences = nltk.sent\_tokenize(text)  sentences = [nltk.word\_tokenize(sentence) for sentence in sentences]  for i in range(len(sentences)):  sentences[i] = [word for word in sentences[i] if word not in stopwords.words('english')]  # Training the Word2Vec model  model = Word2Vec(sentences, min\_count=1)  words = model.wv.vocab  # Finding Word Vectors  vector = model.wv['war']  # Most similar words  similar = model.wv.most\_similar('vikram')  **Implementing Word Embedding using Keras**  ### Libraries USed Tensorflow> 2.0 and keras  ##tensorflow >2.0  from tensorflow.keras.preprocessing.text import one\_hot  *### sentences*  sent=[ 'the glass of milk',  'the glass of juice',  'the cup of tea',  'I am a good boy',  'I am a good developer',  'understand the meaning of words',  'your videos are good',]  ### Vocabulary size  voc\_size=10000  # One Hot Representation  onehot\_repr=[one\_hot(words,voc\_size)for words in sent]  print(onehot\_repr)  **from** **tensorflow.keras.layers** **import** Embedding  **from** **tensorflow.keras.preprocessing.sequence** **import** pad\_sequences  **from** **tensorflow.keras.models** **import** Sequential  **import** **numpy** **as** **np**  sent\_length=8  embedded\_docs=pad\_sequences(onehot\_repr,padding='pre',maxlen=sent\_length)  print(embedded\_docs)  [[ 0 0 0 0 6654 998 8966 1609]  [ 0 0 0 0 6654 998 8966 1428]  [ 0 0 0 0 6654 4519 8966 4736]  [ 0 0 0 8807 9358 6927 7257 6349]  [ 0 0 0 8807 9358 6927 7257 801]  [ 0 0 0 5774 6654 257 8966 4241]  [ 0 0 0 0 4044 1506 8798 7257]]  dim=10  model=Sequential()  model.add(Embedding(voc\_size,10,input\_length=sent\_length))  model.compile('adam','mse')  model.summary()  Model: "sequential\_10"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  embedding\_9 (Embedding) (None, 8, 10) 100000  =================================================================  Total params: 100,000  Trainable params: 100,000  Non-trainable params: 0  print(model.predict(embedded\_docs))  embedded\_docs[0]  print(model.predict(embedded\_docs)[0]) | | |