Word Embedding Tutorial | Word2vec Model Gensim Example

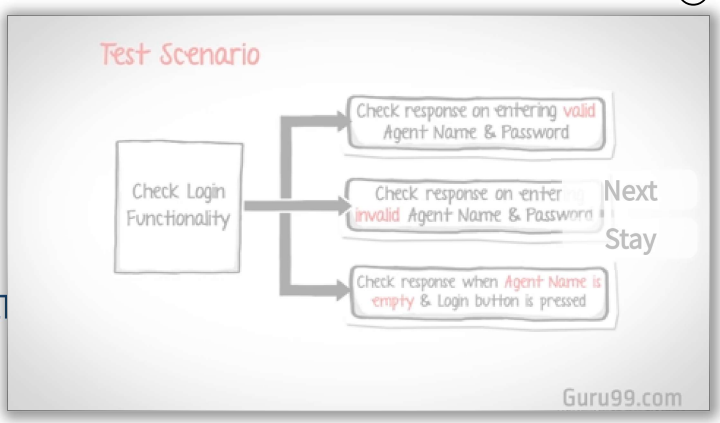
By Daniel Johnson Updated January 1, 2022

What is Word Embedding?

Word Embedding is a word representation type that allows machine learning algorithms to understand words with similar meanings. It is a language modeling and feature learning technique to map words into vectors of real numbers using neural networks, probabilistic models, or dimension reduction on the word co-occurrence matrix. Some word embedding models are Word2vec (Google), Glove (Stanford), and fastest (Facebook).

Word Embedding is also called as distributed semantic model or distributed represented or semantic vector space or vector space model. As you read these names, you come across the word semantic which means categorizing similar words together. For example fruits like apple, mango, banana should be placed close whereas books will be far away from these words. In a broader sense, word embedding will create the vector of fruits which will be placed far away from vector representation of books.

In this Word Embedding tutorial, you will learn:



- What is Gensim?
- How to Implement Word2vec using Gensim

Where is Word Embedding used?

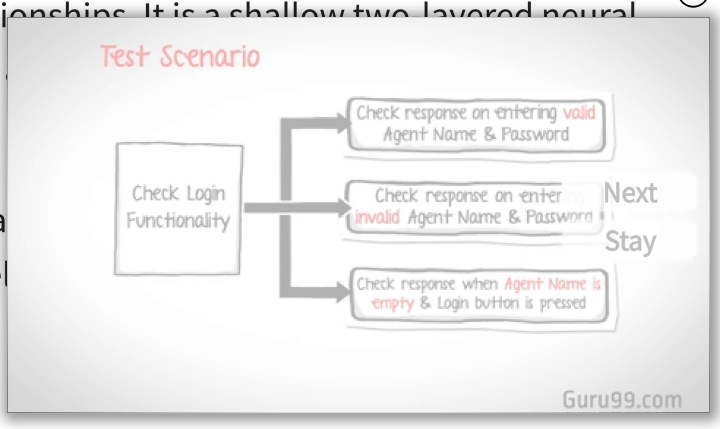
Word embedding helps in feature generation, document clustering, text classification, and natural language processing tasks. Let us list them and have some discussion on each of these applications.

- **Compute similar words:** Word embedding is used to suggest similar words to the word being subjected to the prediction model. Along with that it also suggests dissimilar words, as well as most common words.
- **Create a group of related words:** It is used for semantic grouping which will group things of similar characteristic together and dissimilar far away.
- **Feature for text classification:** Text is mapped into arrays of vectors which is fed to the model for training as well as prediction. Text-based classifier models cannot be trained on the string, so this will convert the text into machine trainable form. Further its features of building semantic help in text-based classification.
- **Document clustering:** is another application where Word Embedding Word2vec is widely used
- **Natural language processing:** There are many applications where word embedding is useful and wins over feature extraction phases such as parts of speech tagging, sentimental analysis, and syntactic analysis.Now we have got some knowledge of word embedding. Some light is also thrown on different models to implement word embedding. This whole Word Embedding tutorial is focused on one of the models (Word2vec).

What is Word2vec?

Word2vec is a technique/model to produce word embedding for better word representation. It is a natural language processing method that captures a large number of precise syntactic and semantic word relati
network that can detect synonymous words
sentences once it is trained.

Before going further in this Word2vec tutoria
and deep neural network as shown in the bel



The shallow neural network consists of the only a hidden layer between input and output whereas deep neural network contains multiple hidden layers between input and output. Input is subjected to nodes whereas the hidden layer, as well as the output layer, contains neurons.

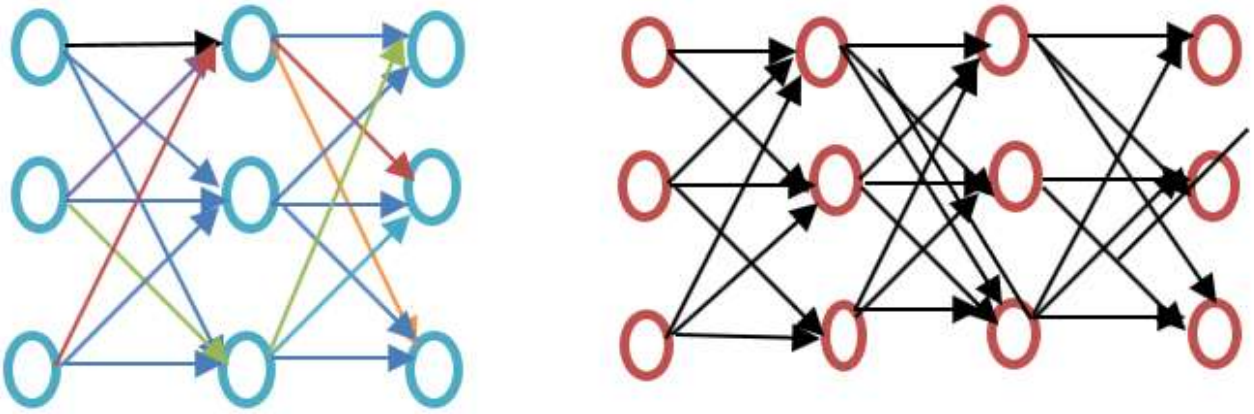


Figure: Shallow vs. Deep learning

Word2vec is a two-layer network where there is input one hidden layer and output.

Word2vec was developed by a group of researcher headed by Tomas Mikolov at Google. Word2vec is better and more efficient that latent semantic analysis model.

Why Word2vec?

Word2vec represents words in vector space r
form of vectors and placement is done in suc
together and dissimilar words are located far
relationship. Neural networks do not underst
numbers. Word Embedding provides a way t

communicate, other people try to figure out what is objective of the sentence. For example, “What is the temperature of India”, here the context is the user wants to know “temperature of India” which is context. In short, the main objective of a sentence is context. Word or sentence surrounding spoken or written language (disclosure) helps in determining the meaning of context. Word2vec learns vector representation of words through the contexts.

What Word2vec does?

Before Word Embedding

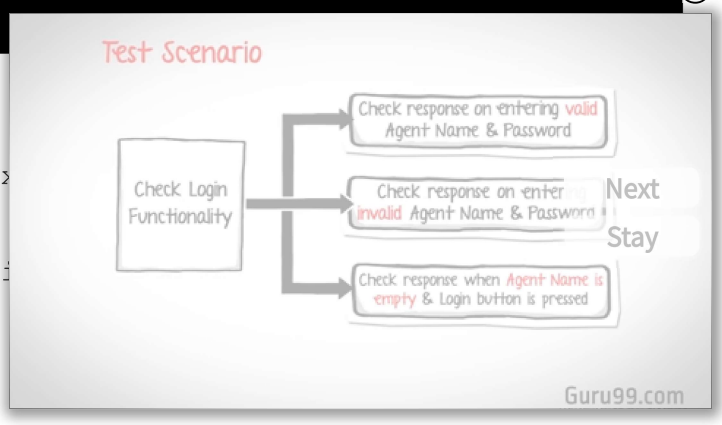
It is important to know which approach is used before word embedding and what are its demerits and then we will move to the topic of how demerits are overcome by Word embedding using Word2vec approach. Finally, we will move how Word2vec works because it is important to understand it’s working.

Approach for Latent Semantic Analysis

This is the approach which was used before word embeddings. It used the concept of Bag of words where words are represented in the form of encoded vectors. It is a sparse vector representation where the dimension is equal to the size of vocabulary. If the word occurs in the dictionary, it is counted, else not. To understand more, please see the below program.

Word2vec Example

```
>>> from sklearn.feature_extraction.text import CountVectorizer 1
>>> vectorizer = CountVectorizer() 2
>>> data_corpus = ["guru99 is the best site for online tutorials. I love to visi
t guru99 ."] 3
>>> vocabulary = vectorizer.fit(data_corpus) 4
>>> X = vectorizer.transform(data_corpus) 5
>>> print(X.toarray()) 6
[[1 1 2 1 1 1 1 1 1 1 1]]
>>> print(vocabulary.get_feature_names()) 7
[u'best', u'for', u'guru99', u'is', u'love', u'online', u'site', u'the', u'to',
u'tutorials', u'visit']
>>>
```




```
from sklearn.feature_extraction.tex
vectorizer=CountVectorizer()
data_corpus=["guru99 is the best si
visit guru99."]
```

```
X= vectorizer.transform(data_corpus)
print(X.toarray())
print(vocabulary.get_feature_names())
```

Output:

```
[[1 2 1 1 1 1 1 1 1 1]]
```

```
[u'best', u'guru99', u'is', u'love', u'online', u'sitefor', u'the', u'to', u'tutorials', u'visit']
```

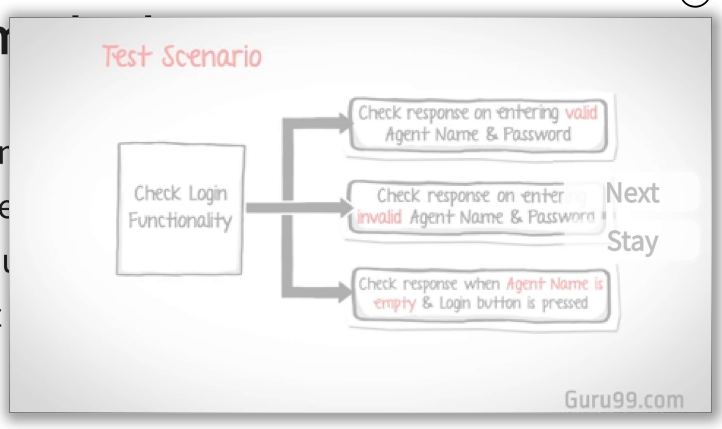
Code Explanation

1. CountVectorizer is the module which is used to store the vocabulary based on fitting the words in it. This is imported from the sklearn.
2. Make the object using the class CountVectorizer.
3. Write the data in the list which is to be fitted in the CountVectorizer.
4. Data is fit in the object created from the class CountVectorizer.
5. Apply a bag of word approach to count words in the data using vocabulary. If word or token is not available in the vocabulary, then such index position is set to zero.
6. Variable in line 5 which is x is converted to an array (method available for x). This will provide the count of each token in the sentence or list provided in Line 3.
7. This will show the features which are part of the vocabulary when it is fitted using the data in Line 4.

In Latent Semantic approach, the row represents unique words whereas the column represents the number of time that word appears in the document. It is a representation of words in the form of the document matrix. Term-Frequency inverse document frequency (TFIDF) is used to count the frequency of words in the document which is the frequency of the term in the document/ frequency of the term in the entire corpus.

Shortcoming of Bag of Words m

- It ignores the order of the word, for exar
- It ignores the context of words. Suppose Education is best found in books". It wou books" and other for "Education is best



orthogonal which makes them independent, but in reality, they are related to each other

To overcome these limitation word embedding was developed and Word2vec is an approach to implement such.

How Word2vec works?

Word2vec learns word by predicting its surrounding context. For example, let us take the word "He **loves** Football."

We want to calculate the Word2vec for the word: loves.

Suppose

```
loves =  V_in. P(V_out / V_in) is calculated
where,
V_in is the input word.
P is the probability of likelihood.
V_out is the output word.
```

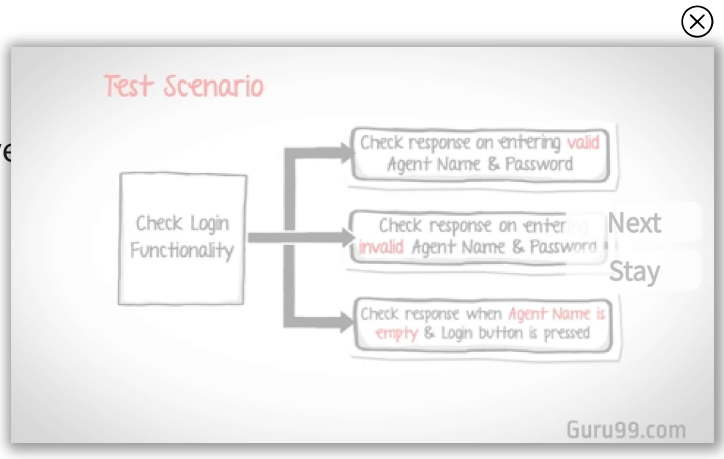
Word **loves** moves over each word in the corpus. Syntactic as well as the Semantic relationship between words is encoded. This helps in finding similar and analogies words.

All random features of the word **loves** is calculated. These features are changed or update concerning neighbor or context words with the help of a Back Propagation method.

Another way of learning is that if the context of two words are similar or two words have similar features, then such words are related.

Word2vec Architecture

There are two architectures used by Word2ve



1. **Continuous Bag of words (CBOW)**
2. **Skip gram**

Before going further in this Word2vec tutorial, let us discuss why these architectures or models are important from word representation point of view. Learning word representation is essentially unsupervised, but targets/labels are needed to train the model. Skip-gram and CBOW convert unsupervised representation to supervised form for model training.

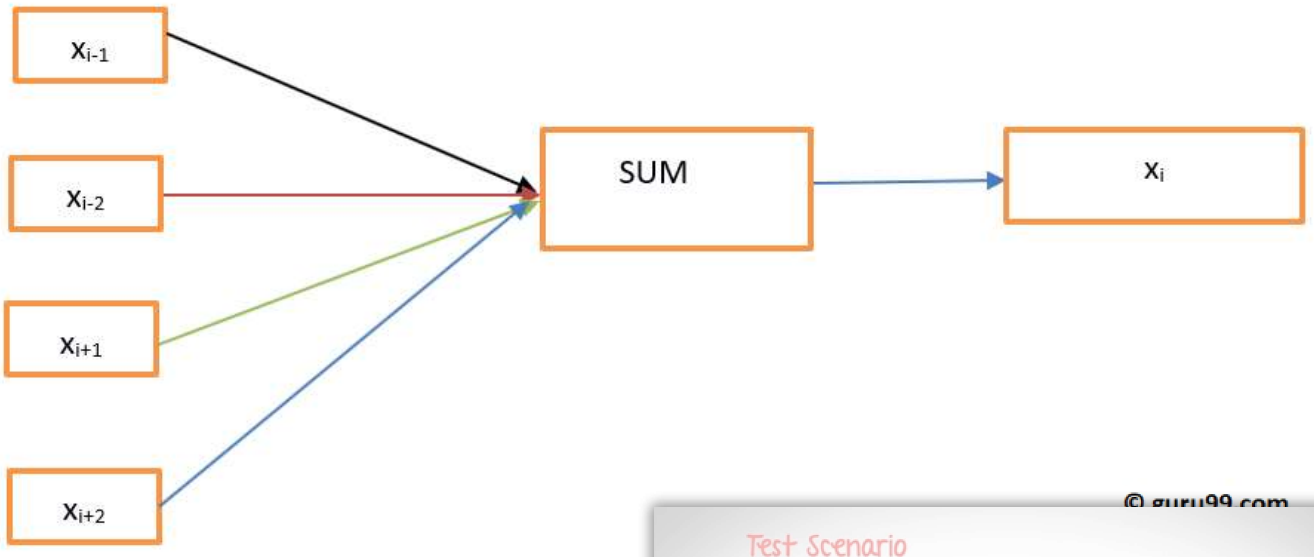
In CBOW, the current word is predicted using the window of surrounding context windows. For example, if $w_{i-1},w_{i-2},w_{i+1},w_{i+2}$are given words or context, this model will provide w_i

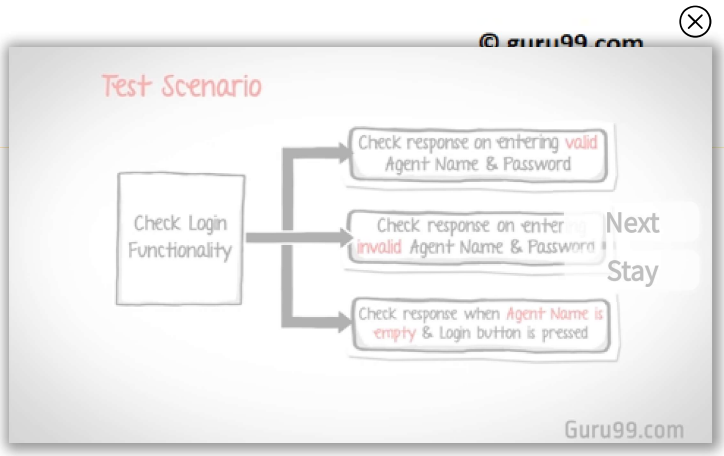
Skip-Gram performs opposite of CBOW which implies that it predicts the given sequence or context from the word. You can reverse the example to understand it. If w_i is given, this will predict the context or $w_{i-1},w_{i-2},w_{i+1},w_{i+2}$.

Word2vec provides an option to choose between CBOW (continuous Bag of words) and skim-gram. Such parameters are provided during training of the model. One can have the option of using negative sampling or hierarchical softmax layer.

Continuous Bag of Words

Let us draw a simple Word2vec example diagram to understand the continuous bag of word architecture.





Let us calculate the equations mathematically. Suppose V is the vocabulary size and N is the hidden layer size. Input is defined as $\{ x_{i-1}, x_{i-2}, x_{i+1}, x_{i+2}\}$. We obtain the weight matrix by multiplying V * N. Another matrix is obtained by multiplying input vector with the weight matrix. This can also be understood by the following equation.

$h=xi^tW$

where xi^t? W are the input vector and weight matrix respectively,

To calculate the match between context and the next word, please refer to the below equation

u=predictedrepresentation*h

where predictedrepresentation is obtained model?h in the above equation.

Skip-Gram Model

Skip-Gram approach is used to predict a sentence given an input word. To understand it better let us draw the diagram as shown in the below Word2vec example.

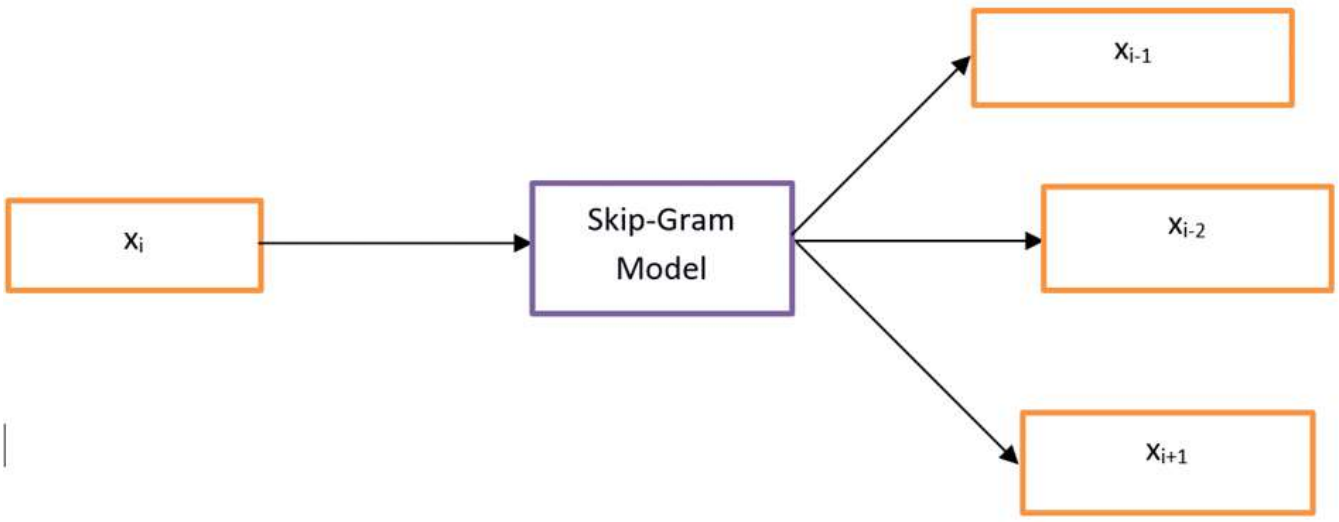
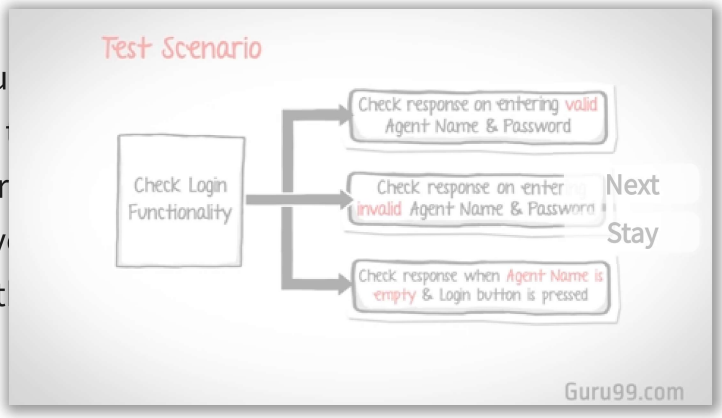


Figure Skip-Gram Model

One can treat it as the reverse of the Continu
the word and model provides the context or
target is fed to the input and output layer is
the chosen number of context words. Error v
to adjust weights via a backpropagation met



CBOW is several times faster than skip gram and provides a better frequency for frequent words whereas skip gram needs a small amount of training data and represents even rare words or phrases.

The relation between Word2vec and NLTK

NLTK is natural Language toolkit. It is used for preprocessing of the text. One can do different operations such as parts of speech tagging, lemmatizing, stemming, stop words removal, removing rare words or least used words. It helps in cleaning the text as well as helps in preparing the features from the effective words. In the other way, Word2vec is used for semantic (closely related items together) and syntactic (sequence) matching. Using Word2vec, one can find similar words, dissimilar words, dimensional reduction, and many others. Another important feature of Word2vec is to convert the higher dimensional representation of the text into lower dimensional of vectors.

Where to use NLTK and Word2vec?

If one has to accomplish some general-purpose tasks as mentioned above like tokenization, POS tagging and parsing one must go for using NLTK whereas for predicting words according to some context, topic modeling, or document similarity one must use Word2vec.

Relation of NLTK and Word2vec with the help of code

NLTK and Word2vec can be used together to find similar words representation or syntactic matching. NLTK toolkit can be used to load many packages which come with NLTK and model can be created using Word2vec. It can be then tested on the real time words. Let us see the combination of both in the following code. Before processing further, please have a look on the corpora which NLTK provides. You can download using the command

```
nltk(nltk.download('all'))
```

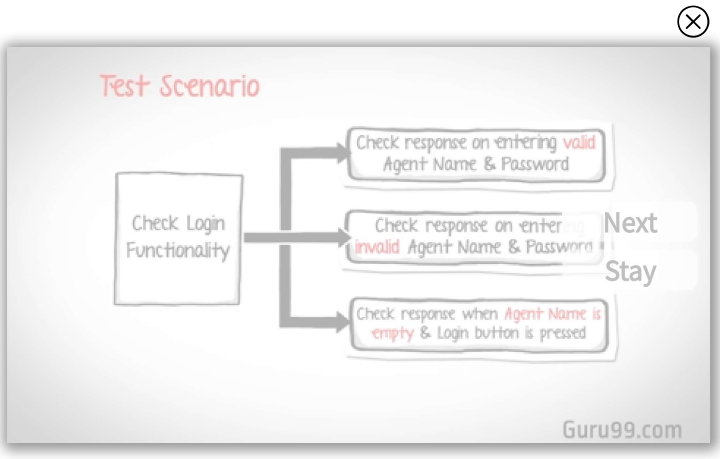


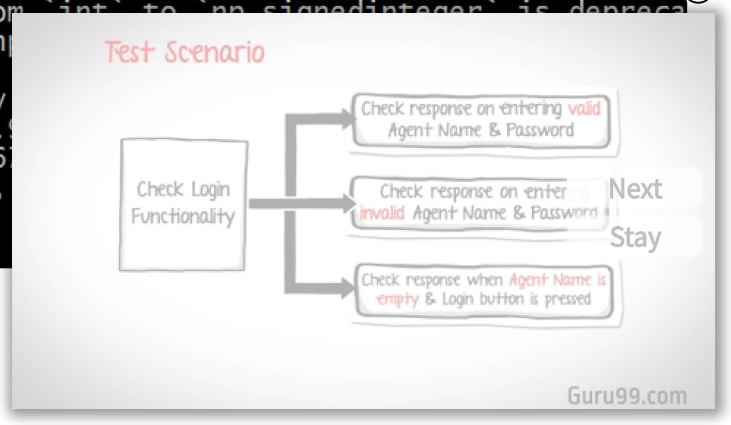
Figure Corpora downloaded using NLTK

Please see the screenshot for the code.

```
import nltk
import gensim
from nltk.corpus import abc

model= gensim.models.Word2Vec(abc.sents())
X= list(model.wv.vocab)
data=model.most_similar('science')
print(data)
```

```
>> import nltk
>> import gensim
:\Python27\lib\site-packages\gensim\utils.py:1212: UserWarning: detected Window
; aliasing chunkize to chunkize_serial
 warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
>> from nltk.corpus import abc
>> model =gensim.models.Word2Vec(abc.sents())
>> X= list(model.wv.vocab)
>> model.most_similar('science')
:\Python27\lib\site-packages\gensim\matutils.py:737: FutureWarning: Conversion
f the second argument of issubdtype from `int` to `np.signedinteger` is deprec
ed. In future, it will be treated as `np
 if np.issubdtype(vec.dtype, np.int):
(u'law', 0.9378671646118164), (u'policy
0.9215561151504517), (u'discussion', 0.
4246216), (u'practice', 0.91201043128961
u'sam', 0.9080902338027954), (u'tight',
65686035156)]
>>
```



Output:

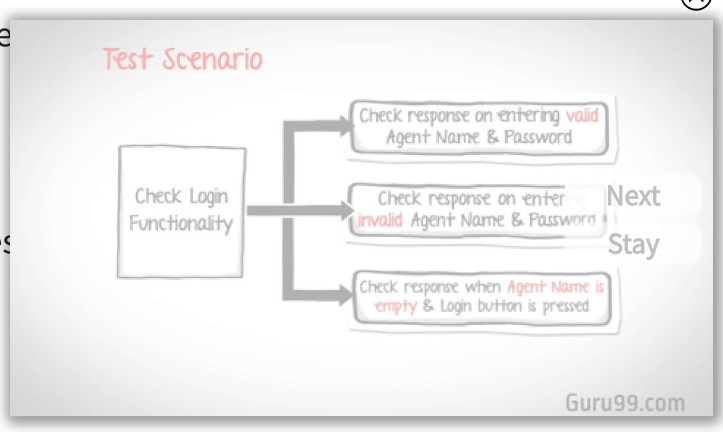
```
[('law', 0.9415997266769409), ('practice', 0.9276568293571472),
('discussion', 0.9259148836135864), ('agriculture',
0.9257254004478455), ('media', 0.9232194423675537), ('policy',
0.922248125076294), ('general', 0.9166069030761719), ('undertaking',
0.916458249092102), ('tight', 0.9129181504249573), ('board',
0.9107444286346436)]
```

Explanation of Code

1. nltk library is imported which from where you can download the abc corpus which we will use in the next step.
2. Gensim is imported. If Gensim Word2vec is not installed, please install it using the command " pip3 install gensim". Please see the below screenshot.

Figure Installing Gensim using PIP

3. import the corpus abc which has been downloaded using nltk.download('abc').
4. Pass the files to the model Word2vec which is imported using Gensim as sentences.
5. Vocabulary is stored in the form of the variable.
6. Model is tested on sample word science as these files are related to science.
7. Here the similar word of "science" is pre



Activators and Word2Vec

The activation function of the neuron defines
inputs. Biologically inspired by an activity in

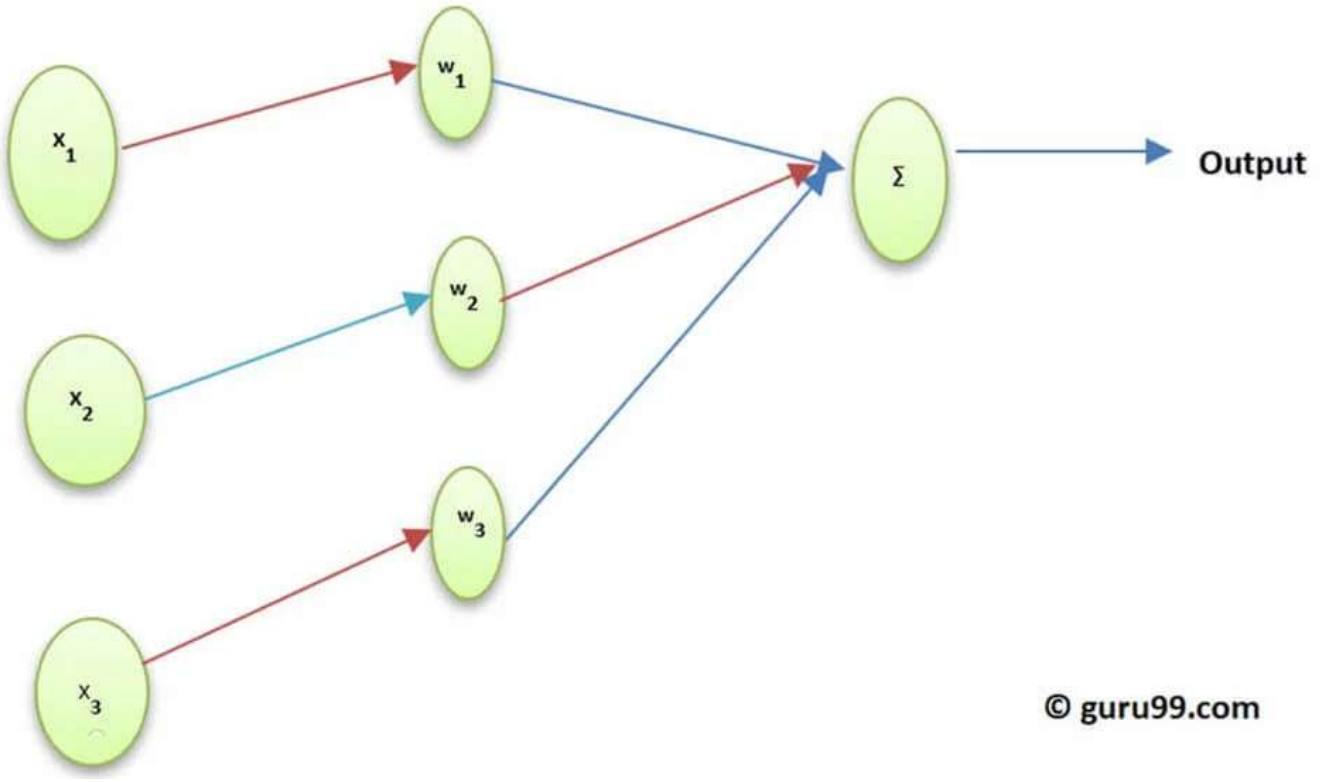


Figure Understanding Activation function

Here x1,x2,..x4 is the node of the neural network.

w1, w2, w3 is the weight of the node,

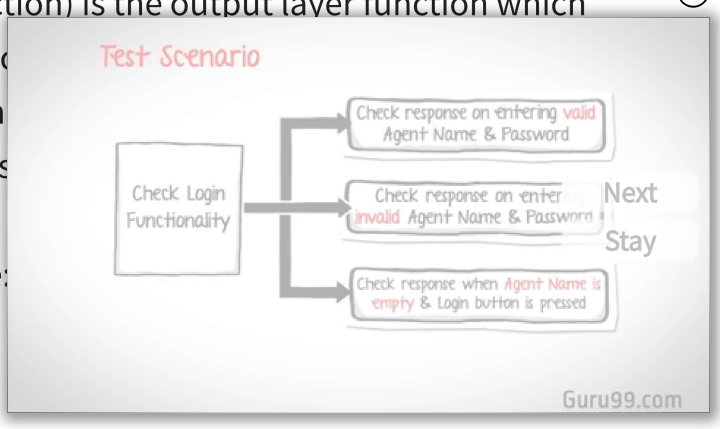
? is the summation of all weight and node value which work as the activation function.

Why Activation function?

If no activation function is used output would be linear but the functionality of linear function is limited. To achieve complex functionality such as object detection, image classification, typing text using voice and many other non-linear outputs is needed which is achieved using activation function.

How the activation layer is computed in the word embedding (Word2vec)

Softmax Layer (normalized exponential function) is the output layer function which activates or fires each node. Another approac
complexity is calculated by $O(\log_2 V)$ wherein
vocabulary size. The difference between thes
hierarchical softmax layer. To understand its
look at the below Word embedding example:



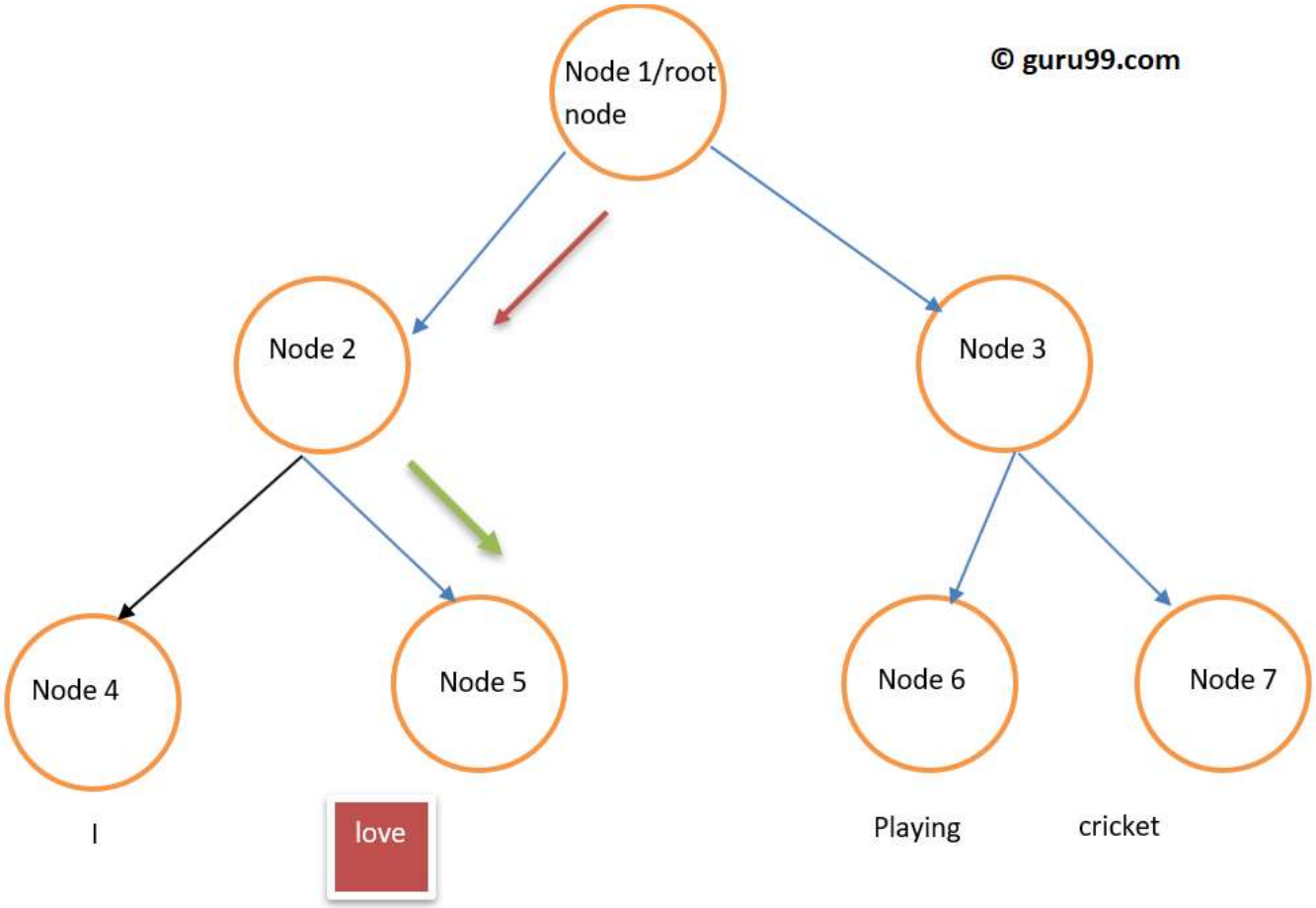


Figure Hierarchical softmax tree like structure

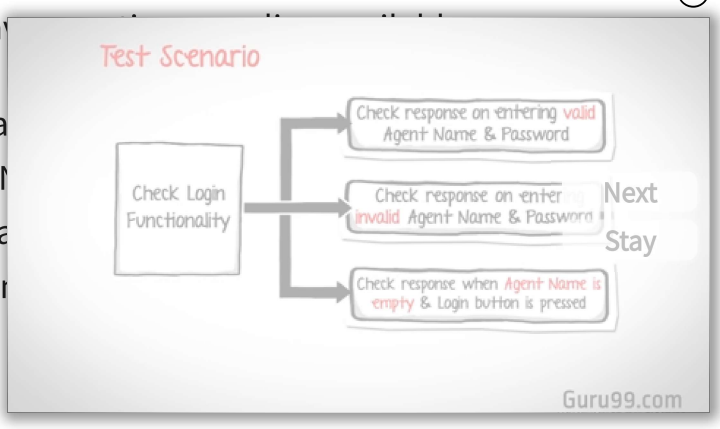
Suppose we want to compute the probability of observing the word **love** given a certain context. The flow from the root to the leaf node will be the first move to node 2 and then to node 5. So if we have had the vocabulary size of 8, only three computations are needed. So it allows decomposing, calculation of the probability of one word (**love**).

What other options are available other than Hierarchical Softmax?

If speaking in a general sense for word embedding options available are Differentiated Softmax, CNN-Softmax, Importance Sampling, Adaptive Importance sampling, Noise Contrastive Estimations, Negative Sampling, Self-Normalization, and infrequent Normalization.

Speaking specifically about Word2vec we ha

Negative Sampling is a way to sample the tra
gradient descent, but with some difference.
training examples. It is based on noise contra
words, not in the context. It is a fast training



What conclusion can be drawn?

Activators are firing the neurons just like our neurons are fired using the external stimuli. Softmax layer is one of the output layer function which fires the neurons in case of word embeddings. In Word2vec we have options such as hierarchical softmax and negative sampling. Using activators, one can convert the linear function into the nonlinear function, and a complex machine learning algorithm can be implemented using such.

What is Gensim?

Gensim is an open-source topic modeling and natural language processing toolkit that is implemented in Python and Cython. Gensim toolkit allows users to import Word2vec for topic modeling to discover hidden structure in the text body. Gensim provides not only an implementation of Word2vec but also for Doc2vec and FastText as well.

This tutorial is all about Word2vec so we will stick to the current topic.

How to Implement Word2vec using Gensim

Till now we have discussed what Word2vec is, its different architectures, why there is a shift from a bag of words to Word2vec, the relation between Word2vec and NLTK with live code and activation functions.

Below is the step by step method to implement Word2vec using Gensim:

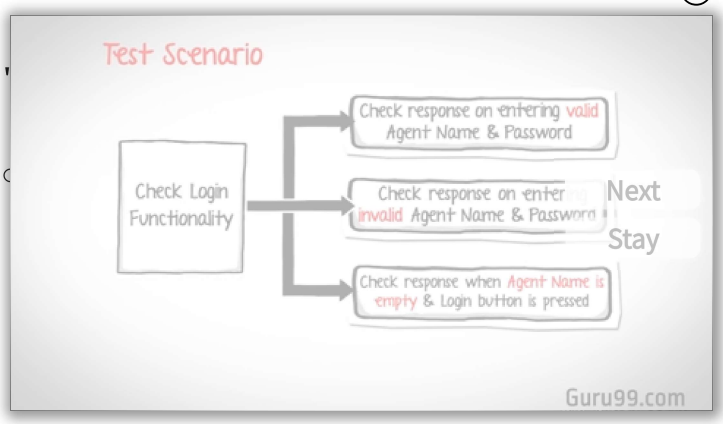
Step 1) Data Collection

The first Step to implement any machine learning model or implementing natural language processing is data collection

Please observe the data to build an intelligent chatbot as shown in the below Gensim Word2vec example.

```
[{"tag": "welcome",
"patterns": ["Hi", "How are you", "
are you available"],
"responses": ["Hello, thanks for co
here"," Hi there, how may I assist

        },
```



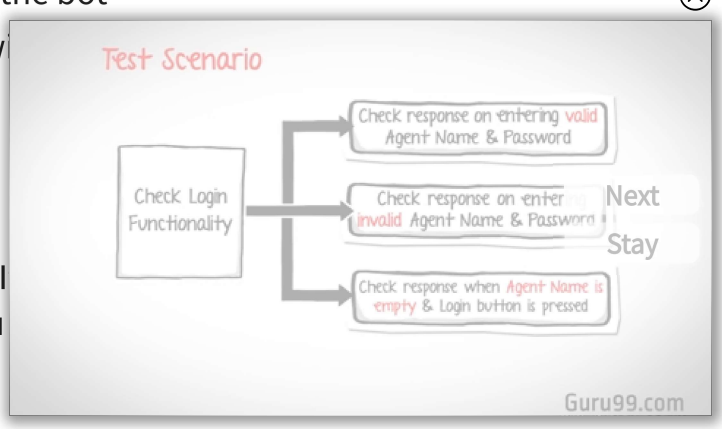
```
"patterns": ["Bye", "See you later", "Goodbye", "I will come back
soon"],
"responses": ["See you later, thanks for visiting", "have a great day
ahead", "Wish you Come back again soon."]
        },

{"tag": "thankful",
"patterns": ["Thanks for helping me", "Thank your guidance", "That's
helpful and kind from you"],
"responses": ["Happy to help!", "Any time!", "My pleasure", "It is my
duty to help you"]
        },
        {"tag": "hoursopening",
"patterns": ["What hours are you open?", "Tell your opening time?",
"When are you open?", "Just your timing please"],
"responses": ["We're open every day 8am-7pm", "Our office hours are
8am-7pm every day", "We open office at 8 am and close at 7 pm"]
        },

{"tag": "payments",
"patterns": ["Can I pay using credit card?", " Can I pay using
Mastercard?", " Can I pay using cash only?" ],
"responses": ["We accept VISA, Mastercard and credit card", "We
accept credit card, debit cards and cash. Please don't worry"]
        }
   ]
```

Here is what we understand from the data

- This data contains three things tag, pattern, and responses. The tag is the intent (what is the topic of discussion).
- The data is in JSON format.
- A pattern is a question users will ask to the bot
- Responses is the answer that chatbot w
  question/pattern.



Step 2) Data preprocessing

It is very important to process the raw data. I
model will respond more accurately and will

This step involves removing stop words, stemming, unnecessary words, etc. Before going ahead, it is important to load data and convert it into a data frame. Please see the below code for such

```
import json
json_file ='intents.json'
with open('intents.json','r') as f:
    data = json.load(f)
```

Explanation of Code:

1. As data is in the form of json format hence json is imported
2. File is stored in the variable
3. File is open and loaded in data variable

Now data is imported and it is time to convert data into data frame. Please see the below code to see the next step

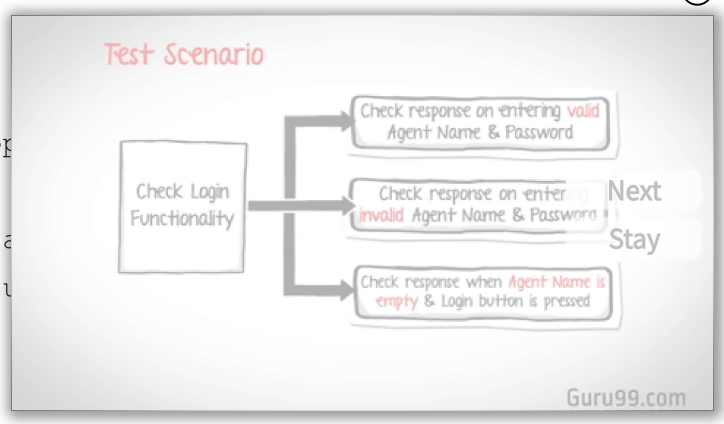
```
import pandas as pd
df = pd.DataFrame(data)
df['patterns'] = df['patterns'].apply(', '.join)
```

Explanation of Code:

1. Data is converted into data frame using pandas which was imported above.

2. It will convert the list in column patterns to string.

```
from nltk.corpus import stopwords
from textblob import Word
stop = stopwords.words('english')
df['patterns'] = df['patterns'].app
x in x.split()))
df['patterns't']= df['patterns''].a
x.split() if x not in string.punctu
```



```
x.split() if  not x.isdigit()))
df['patterns'] = df['patterns'].apply(lambda x:' '.join(x for x in
x.split() if not x in stop))
df['patterns'] = df['patterns'].apply(lambda x: "
".join([Word(word).lemmatize() for word in x.split()]))
```

Code Explanation:

1. English stop words are imported using stop word module from nltk toolkit

2. All the words of the text is converted into lower case using for condition and lambda function. Lambda function is an anonymous function.

3. All the rows of the text in the data frame is checked for string punctuations, and these are filtered.

4. Characters such as numbers or dot are removed using a regular expression.

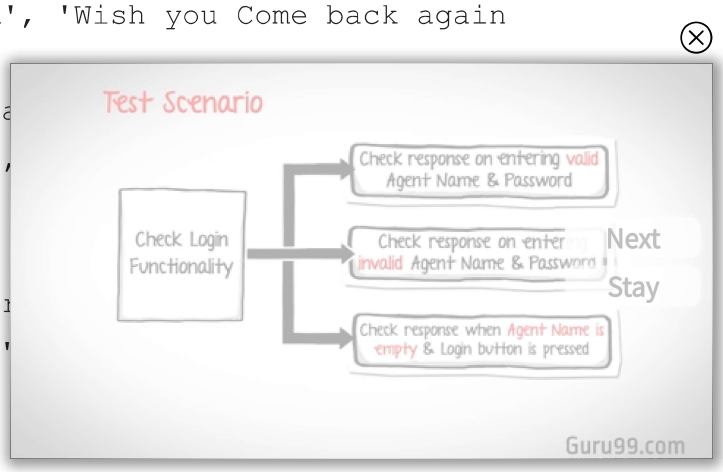
5. Digits are removed from the text.

6. Stop words are removed at this stage.

7. Words are filtered now, and different form of the same word is removed using lemmatization. With these, we have finished the data preprocessing.

Output:

```
, patterns, responses, tag
0,hi one talk hello hi available,"['Hello, thanks for contacting us',
'Good to see you here', ' Hi there, how may I assist you?']",welcome
1,bye see later goodbye come back soon,"['See you later, thanks for
visiting', 'have a great day ahead', 'Wish you Come back again
soon.']",goodbye
2,thanks helping thank guidance tha
help!', 'Any time!', 'My pleasure',
you']",thankful
3,hour open tell opening time open
day 8am-7pm"", 'Our office hours ar
office at 8 am and close at 7 pm']'
```



```
accept VISA, Mastercard and credit card', 'We accept credit card,
debit cards and cash. Please donâ€™t worry']",payments
```

Step 3) Neural Network building using Word2vec

Now it is time to build a model using Gensim Word2vec module. We have to import Word2vec from Gensim. Let us do this, and then we will build and in the final stage we will check the model on real time data.

```
from gensim.models import Word2Vec
```

Now in this Gensim Word2vec tutorial, we can successfully build the model using Word2Vec. Please refer to the next line of code to learn how to create the model using Word2Vec. Text is provided to the model in the form of a list so we will convert the text from data frame to list using the below code

```
Bigger_list=[]
for i in df['patterns']
     li = list(i.split(""))
     Bigger_list.append(li)
Model= Word2Vec(Bigger_list,min_count=1,size=300,workers=4)
```

Explanation of Code:

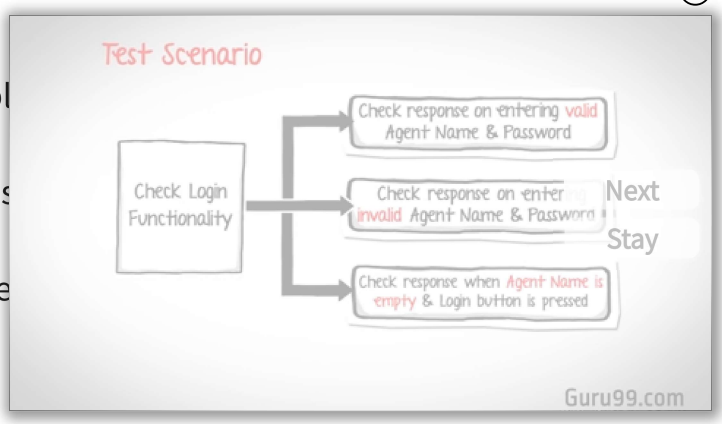
1. Created the bigger_list where the inner list is appended. This is the format which is fed to the model Word2Vec.

2. Loop is implemented, and each entry of the patterns column of the data frame is iterated.

3. Each element of the column patterns is spl

4. the Inner list is appended with the outer lis

5. This list is provided to the Word2Vec mode



Min_count: It will ignore all the words with a total frequency lower than this.

Size: It tells the dimensionality of the word vectors.

Workers: These are the threads to train the model

There are also others options available, and some important ones are explained below

Window: Maximum distance between the current and predicted word within a sentence.

Sg: It is a training algorithm and 1 for skip-gram and 0 for a Continuous bag of words. We have discussed these in details in above.

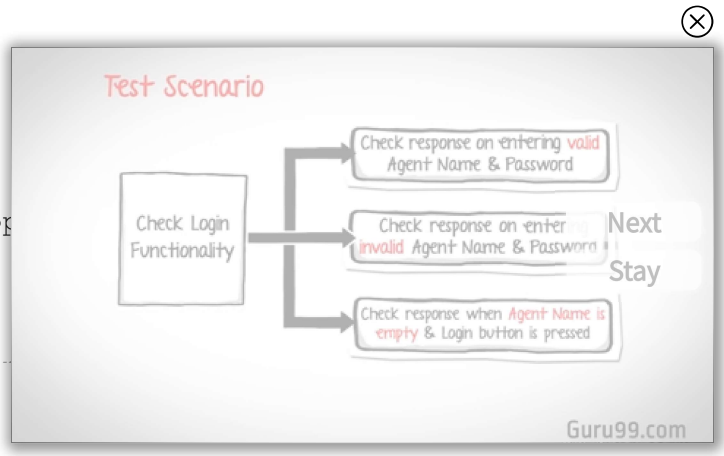
Hs: If this is 1 then we are using hierarchical softmax for training and if 0 then negative sampling is used.

Alpha: Initial learning rate

Let us display the final code below:

```
#list of libraries used by the code
import string
from gensim.models import Word2Vec
import logging
from nltk.corpus import stopwords
from textblob import Word
import json
import pandas as pd
#data in json format
json_file = 'intents.json'
with open('intents.json','r') as f:
    data = json.load(f)
#displaying the list of stopwords
stop = stopwords.words('english')
#dataframe
df = pd.DataFrame(data)

df['patterns'] = df['patterns'].app
# print(df['patterns'])
#print(df['patterns'])
```



```
df['patterns'] = df['patterns'].apply(lambda x:' '.join(x.lower() for
x in x.split()))
df['patterns']= df['patterns'].apply(lambda x: ' '.join(x for x in
x.split() if x not in string.punctuation))
df['patterns']= df['patterns'].str.replace('[^\w\s]','')
df['patterns']= df['patterns'].apply(lambda x: ' '.join(x for x in
x.split() if  not x.isdigit()))
df['patterns'] = df['patterns'].apply(lambda x:' '.join(x for x in
x.split() if not x in stop))
df['patterns'] = df['patterns'].apply(lambda x: "
".join([Word(word).lemmatize() for word in x.split()]))
#taking the outer list
bigger_list=[]
for i in df['patterns']:
    li = list(i.split(" "))
    bigger_list.append(li)
#structure of data to be taken by the model.word2vec
print("Data format for the overall list:",bigger_list)
#custom data is fed to machine for further processing
model = Word2Vec(bigger_list, min_count=1,size=300,workers=4)
#print(model)
```

Step 4) Model saving

Model can be saved in the form of bin and model form. Bin is the binary format. Please see the below lines to save the model

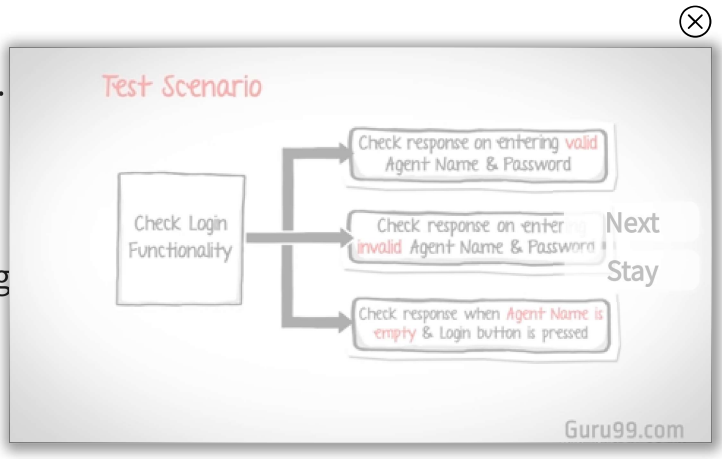
```
model.save("word2vec.model")
model.save("model.bin")
```

Explanation of the above code

1. Model is saved in the form of a .model file.

2. model is saved in the form of .bin file

We will use this model to do real time testing most common words.



Model is loaded using below code:

```
model = Word2Vec.load('model.bin')
```

If you want to print the vocabulary from it is done using below command:

```
vocab = list(model.wv.vocab)
```

Please see the result:

```
['see', 'thank', 'back', 'thanks', 'soon', 'open', 'mastercard',
'card', 'time', 'pay', 'talk', 'cash', 'one', 'please', 'goodbye',
'thats', 'helpful', 'hour', 'credit', 'hi', 'later', 'guidance',
'opening', 'timing', 'hello', 'helping', 'bye', 'tell', 'come',
'using', 'kind', 'available']
```

Step 6) Most Similar words checking

Let us implement the things practically:

```
similar_words = model.most_similar('thanks')
print(similar_words)
```

Please see the result:

```
[('kind', 0.16104359924793243), ('using', 0.1352398842573166),
('come', 0.11500970274209976), ('la
('helping', 0.04855936020612717),
('pay', 0.0329081267118454), ('thar
0.0202352125197649), ('opening', 0.
```

```
dissimlar_words = model.doesnt_match('See you later, thanks for
visiting'.split())
print(dissimlar_words)
```

We have supplied the words **'See you later, thanks for visiting'.** This will print the most dissimilar words from these words. Let us run this code and find the result

The result after execution of the above code:

```
Thanks
```

Step 8) Finding the similarity between two words

This will tell result in probability of similarity between two words. Please see the below code how to execute this section.

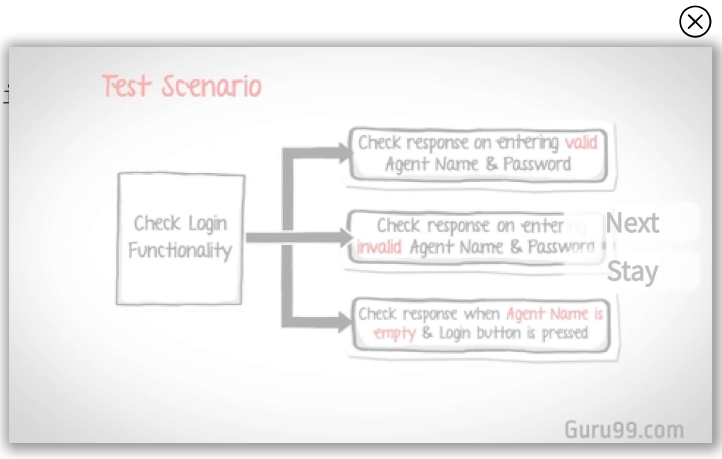
```
similarity_two_words = model.similarity('please','see')
print("Please provide the similarity between these two words:")
print(similarity_two_words)
```

The result of the above code is as below

```
0.13706
```

You can further find similar words by executing the below code

```
similar = model.similar_by_word('ki
print(similar)
```



Output of above code:

```
[('credit', 0.11764447391033173), ('cash', 0.11440904438495636),
('one', 0.11151769757270813), ('hour', 0.0944807156920433), ('using',
0.0705675333738327), ('thats', 0.05206916481256485), ('later',
0.04502468928694725), ('bye', 0.03960943967103958), ('back',
0.03837274760007858), ('thank', 0.0380823090672493)]
```

Conclusion

- Word Embedding is a type of word representation that allows words with similar meaning to be understood by machine learning algorithms
- Word Embedding is used to compute similar words, Create a group of related words, Feature for text classification, Document clustering, Natural language processing
- Word2vec explained: Word2vec is a shallow two-layered neural network model to produce word embeddings for better word representation
- Word2vec represents words in vector space representation. Words are represented in the form of vectors and placement is done in such a way that similar meaning words appear together and dissimilar words are located far away
- Word2vec algorithm uses 2 architectures Continuous Bag of words (CBOW) and skip gram
- CBOW is several times faster than skip gram and provides a better frequency for frequent words whereas skip gram needs a small amount of training data and represents even rare words or phrases.
- NLTK and Word2vec can be used together create powerful applications
- The activation function of the neuron defines the output of that neuron given a set of inputs. In Word2vec. Softmax Layer (normalized exponential function) is the output layer function which activates or fires each node. Word2vec also has negative sampling available
- Gensim is a topic modeling toolkit which is implemented in python

You Might Like:

- How to Download & Install NLTK on Win
- NLTK Tokenize: Words and Sentences To
- POS Tagging with NLTK and Chunking in
- Stemming and Lemmatization in Python

About

About Us
Advertise with Us
Write For Us
Contact Us

Career Suggestion

SAP Career Suggestion Tool
Software Testing as a Career

Interesting

eBook
Blog
Quiz
SAP eBook

Execute online

Execute Java Online
Execute Javascript
Execute HTML
Execute Python