

#

## ASSIGNMENT 05

Name: Abhinay G. Giri

Roll No: 43127

Class: BE 09

Batch: Q9

Title: Implement the Continuous Bag of Words (CBOW) Model.

```
In [8]: #importing libraries
from keras.preprocessing import text
from keras.utils import np_utils
from keras.preprocessing import sequence
from keras.utils import pad_sequences
import numpy as np
import pandas as pd
```

```
In [9]: #taking random sentences as data
data = """Deep learning (also known as deep structured learning) is part
Deep-learning architectures such as deep neural networks, deep belief ne
"""
dl_data = data.split()
```

```
In [13]: #tokenization
tokenizer = text.Tokenizer()
tokenizer.fit_on_texts(dl_data)
word2id = tokenizer.word_index

word2id['PAD'] = 0
id2word = {v:k for k, v in word2id.items()}
wids = [[word2id[w] for w in text.text_to_word_sequence(doc)] for doc in

vocab_size = len(word2id)
embed_size = 100
window_size = 2

print('Vocabulary Size:', vocab_size)
print('Vocabulary Sample:', list(word2id.items())[:10])
```

Vocabulary Size: 75

Vocabulary Sample: [('learning', 1), ('deep', 2), ('networks', 3), ('neural', 4), ('and', 5), ('as', 6), ('of', 7), ('machine', 8), ('supervised', 9), ('have', 10)]

```

In [18]: #generating (context word, target/label word) pairs
def generate_context_word_pairs(corpus, window_size, vocab_size):
    context_length = window_size*2
    for words in corpus:
        sentence_length = len(words)
        for index, word in enumerate(words):
            context_words = []
            label_word = []
            start = index - window_size
            end = index + window_size + 1

            context_words.append([words[i]
                                for i in range(start, end)
                                if 0 <= i < sentence_length
                                and i != index])
            label_word.append(word)

            x = pad_sequences(context_words, maxlen=context_length)
            y = np_utils.to_categorical(label_word, vocab_size)
            yield (x, y)

i = 0
for x, y in generate_context_word_pairs(corpus=wids, window_size=window_
if 0 not in x[0]:
    # print('Context (X):', [id2word[w] for w in x[0]], '-> Target

    if i == 10:
        break
    i += 1

```

```
In [19]: #model building
import keras.backend as K
from keras.models import Sequential
from keras.layers import Dense, Embedding, Lambda

cbow = Sequential()
cbow.add(Embedding(input_dim=vocab_size, output_dim=embed_size, input_length=1))
cbow.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=(embed_size,)))
cbow.add(Dense(vocab_size, activation='softmax'))
cbow.compile(loss='categorical_crossentropy', optimizer='rmsprop')

print(cbow.summary())

# from IPython.display import SVG
# from keras.utils.vis_utils import model_to_dot

# SVG(model_to_dot(cbow, show_shapes=True, show_layer_names=False, rankdir='LR'))
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
=====		
embedding_1 (Embedding)	(None, 4, 100)	7500
lambda_1 (Lambda)	(None, 100)	0
dense_1 (Dense)	(None, 75)	7575
=====		
Total params: 15,075		
Trainable params: 15,075		
Non-trainable params: 0		
None		

```
In [4]: for epoch in range(1, 6):
        loss = 0.
        i = 0
        for x, y in generate_context_word_pairs(corpus=wids, window_size=wir):
            i += 1
            loss += cbow.train_on_batch(x, y)
            if i % 100000 == 0:
                print('Processed {} (context, word) pairs'.format(i))

        print('Epoch:', epoch, '\tLoss:', loss)
        print()
```

```
Epoch: 1          Loss: 434.3181896209717
Epoch: 2          Loss: 429.8252649307251
Epoch: 3          Loss: 426.54452538490295
Epoch: 4          Loss: 423.13419938087463
Epoch: 5          Loss: 420.3350956439972
```

```
In [5]: weights = cbow.get_weights()[0]
        weights = weights[1:]
        print(weights.shape)

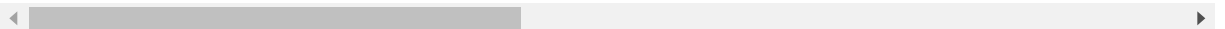
        pd.DataFrame(weights, index=list(id2word.values())[1:]).head()

(74, 100)
```

```
Out[5]:
```

	0	1	2	3	4	5	6	7
<b>deep</b>	0.023335	-0.052239	0.049198	0.017686	0.043500	-0.032212	0.001213	0.021125
<b>networks</b>	-0.025227	-0.036622	0.058194	0.051734	0.024122	-0.012788	-0.040460	0.026885
<b>neural</b>	-0.035517	0.006722	0.010547	0.011032	0.020513	0.016522	-0.024069	0.019897
<b>and</b>	0.007806	-0.032948	0.038503	0.019530	-0.000720	0.044247	-0.015843	-0.015839
<b>as</b>	-0.016440	-0.016150	0.027937	-0.046403	0.022232	0.011129	-0.019134	0.013406

5 rows × 100 columns



```
In [7]: from sklearn.metrics.pairwise import euclidean_distances

distance_matrix = euclidean_distances(weights)
print(distance_matrix.shape)

similar_words = {search_term: [id2word[idx] for idx in distance_matrix[v]
                               for search_term in ['deep']}]

similar_words

(74, 74)
```

```
Out[7]: {'deep': ['learning', 'methods', 'to', 'can', 'in']}
```