

Assignment 5: Implement the Continuous Bag of Words (CBOW) Model

```
In [ ]: #Tushar Kokane  
#B511066 Div: A
```

```
In [1]: #importing libraries  
from keras.preprocessing import text  
from keras.preprocessing import sequence  
from keras.utils import pad_sequences  
from keras.utils import to_categorical  
import numpy as np  
import pandas as pd
```

```
In [2]: #taking random sentences as data  
data = """Deep learning (also known as deep structured learning) is part of  
Deep-learning architectures such as deep neural networks, deep belief netwo  
"""  
dl_data = data.split()
```

```
In [3]: #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 dl_data]  
  
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), ('neura  
l', 4), ('and', 5), ('as', 6), ('of', 7), ('machine', 8), ('supervised',  
9), ('have', 10)]
```

```

In [5]: #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 = to_categorical(label_word, vocab_size)
            yield (x, y)

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

        if i == 10:
            break
        i += 1

```

```
In [6]: #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=
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 4, 100)	7500
lambda (Lambda)	(None, 100)	0
dense (Dense)	(None, 75)	7575
Total params: 15075 (58.89 KB)		
Trainable params: 15075 (58.89 KB)		
Non-trainable params: 0 (0.00 Byte)		
None		

```
In [8]: for epoch in range(1, 6):
        loss = 0.
        i = 0
        for x, y in generate_context_word_pairs(corpus=wids, window_size=window):
            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: 418.98874950408936

Epoch: 2      Loss: 417.880343914032

Epoch: 3      Loss: 417.1407799720764

Epoch: 4      Loss: 416.5916874408722

Epoch: 5      Loss: 416.14987683296204
```

```
In [9]: weights = cbow.get_weights()[0]
weights = weights[1:]
print(weights.shape)

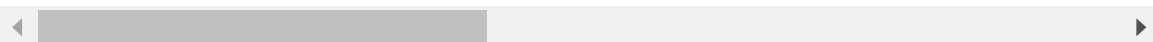
pd.DataFrame(weights, index=list(id2word.values())[1:]).head()
```

(74, 100)

Out[9]:

	0	1	2	3	4	5	6	7
deep	0.008229	-0.072051	0.004338	-0.007478	0.073629	0.037738	0.002554	-0.019888
networks	0.003596	0.006270	-0.005325	-0.000460	0.049438	0.063903	0.070760	0.019638
neural	-0.043509	0.010417	-0.044186	-0.031912	0.036717	-0.018345	0.041918	0.001976
and	-0.007123	0.020363	-0.001633	-0.033472	0.035727	0.020133	0.021322	0.019051
as	0.008709	0.030683	-0.021871	0.039532	0.043744	-0.049553	0.024570	-0.014947

5 rows × 100 columns



In []:

```
In [11]: 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[word
                                for search_term in ['deep']]}

similar_words
```

(74, 74)

Out[11]: {'deep': ['also', 'analysis', 'material', 'human', 'artificial']}