Name: **Pranit Zanwar**

Div: **BE09-S09**

Roll no: **43181**

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

```
#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
```

```
#taking random sentences as data
data = """Deep learning (also known as deep structured learning) is part of a broader family
Deep-learning architectures such as deep neural networks, deep belief networks, deep reinforc
"""
dl_data = data.split()
```

```
#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), ('neural', 4), ('and
```

```
#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_size, vocab_size=voca
    if 0 not in x[0]:
        # print('Context (X):', [id2word[w] for w in x[0]], '-> Target (Y):', id2word[np.argw

        if i == 10:
            break
        i += 1



#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=window_size*2))
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='TB').create(prog=

    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: 15,075
Trainable params: 15,075
Non-trainable params: 0
_________________________________________________________________
None
```

```
for epoch in range(1, 6):
    loss = 0.
    i = 0
    for x, y in generate_context_word_pairs(corpus=wids, window_size=window_size, vocab_size=
        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: 433.88288593292236

Epoch: 2        Loss: 428.80262994766235

Epoch: 3        Loss: 425.27664613723755

Epoch: 4        Loss: 422.0698399543762

Epoch: 5        Loss: 419.70696926116943
```

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

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

```
(74, 100)
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| **deep** | -0.023805 | 0.014639 | 0.007089 | 0.012630 | -0.019672 | -0.024277 | 0.032819 |
| **networks** | 0.037381 | 0.012409 | -0.048280 | 0.010494 | 0.014020 | -0.050136 | -0.056002 |
| **neural** | 0.010798 | -0.013446 | -0.036526 | 0.040533 | -0.028618 | -0.038640 | 0.031019 |
| **and** | -0.042606 | -0.042891 | 0.007063 | -0.033965 | -0.027141 | 0.030406 | -0.049688 |
| **as** | -0.010727 | -0.016468 | -0.048425 | -0.025175 | -0.020269 | 0.023518 | -0.036965 |

5 rows × 100 columns

```
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[word2id[search_term]-1
                  for search_term in ['deep']}

similar_words
```

```
(74, 74)
{'deep': ['analysis',
  'recognition',
  'representation',
  'belief',
  'unsupervised']}
```