```
#importing libraries
from keras.preprocessing import text
from keras.utils import to_categorical
from keras.preprocessing import sequence
from keras.utils import pad_sequences
import numpy as np
import pandas as pd
```

```
C:\Users\ishwa\anaconda3\lib\site-packages\scipy\__init__.py:155:
UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this
version of SciPy (detected version 1.26.1
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
```

```
#taking random sentences as data
data = """Deep learning (also known as deep structured learning) is
part of a broader family of machine learning methods based on
artificial neural networks with representation learning. Learning can
be supervised, semi-supervised or unsupervised.
Deep-learning architectures such as deep neural networks, deep belief
networks, deep reinforcement learning, recurrent neural networks,
convolutional neural networks and Transformers have been applied to
fields including computer vision, speech recognition, natural language
processing, machine translation, bioinformatics, drug design, medical
image analysis, climate science, material inspection and board game
programs, where they have produced results comparable to and in some
cases surpassing human expert performance.
"""
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', 5), ('as', 6), ('of', 7), ('machine', 8),
('supervised', 9), ('have', 10)]

#generating (context word, target/label word) pairs
from keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences
import numpy as np

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

     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='dot', format='svg'))
```

```
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
```

```
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=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: 433.8418712615967

Epoch: 2   Loss: 429.42401337623596

Epoch: 3   Loss: 426.17510986328125

Epoch: 4   Loss: 423.04978919029236

Epoch: 5   Loss: 420.5637102127075
```

```
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   \
deep     -0.000299  0.022343 -0.065450 -0.043762  0.049273  0.051278

networks -0.023542 -0.006513 -0.031387  0.039449 -0.013439  0.018015

neural   -0.024584  0.048279 -0.025915  0.026436 -0.023392  0.011962

and      -0.035015 -0.009195 -0.020977  0.017876  0.035907  0.047437

as        0.041176  0.037102 -0.042300 -0.018578  0.003896  0.019457


                6         7         8         9   ...         90
91  \
deep      0.022365  0.026896 -0.052491  0.009203  ...  0.011724
0.019811
networks  0.002051 -0.028169 -0.004148  0.026132  ...  0.016861
0.009214
neural   -0.040706 -0.012937 -0.019484 -0.010171  ... -0.012715
0.001366
and      -0.001912 -0.036220 -0.038946 -0.009565  ... -0.048995 -
0.022683
as        0.019993 -0.005296 -0.018042  0.031485  ... -0.001497 -
0.016245

                92        93        94        95        96
97  \
deep     -0.000504 -0.014028 -0.051884  0.029571  0.065389  0.017996

networks -0.036904  0.013557  0.001124 -0.065585  0.007065  0.047569

neural    0.043877  0.026237 -0.024337 -0.041876  0.041598  0.022363

and       0.031260 -0.015266  0.034584  0.037140  0.042101 -0.008063

as        0.041384  0.026128 -0.009671 -0.009986  0.037402  0.011527


                98        99
deep     -0.038894  0.003861
networks  0.017940 -0.016235
neural    0.023430 -0.013458
```

```
and       -0.027656 -0.001721
as        -0.028652 -0.013216

[5 rows x 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].argsort()[1:6]+1]
                   for search_term in ['deep']}

similar_words
```

```
(74, 74)

{'deep': ['results', 'translation', 'or', 'image', 'analysis']}
```