```
# 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
```

```
#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 networ
Deep-learning architectures such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks, convolutional neur
"""
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
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.argwhere(y[0])[0][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: 433.3223338127136

Epoch: 2        Loss: 429.139390707016

Epoch: 3        Loss: 425.9573450088501

Epoch: 4        Loss: 422.89519119262695

Epoch: 5        Loss: 420.44101786613464
```

```
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	7	8	9	...	90	91	92	93	
deep	-0.001573	-0.052912	-0.017131	0.002888	0.019808	0.063146	-0.016440	-0.021288	0.051638	0.050375	...	-0.019526	0.057400	-0.053945	-0.050507	0.03
networks	-0.063048	-0.019165	-0.002312	-0.022846	0.027579	0.045859	0.037940	-0.034691	0.026640	-0.003555	...	0.027643	0.018975	-0.003551	0.015731	-0.01
neural	-0.000963	-0.010728	0.048661	-0.018589	0.034974	-0.040866	-0.032136	0.014558	-0.019917	-0.010675	...	-0.037761	-0.019428	0.045281	-0.039951	0.01
and	-0.042171	0.035908	-0.026444	-0.018464	0.008575	-0.035663	0.007724	0.026006	0.020505	0.043628	...	0.007625	0.009627	-0.043330	0.001984	-0.03
as	0.026879	0.014034	0.008110	-0.043944	0.024382	0.003730	0.027463	0.021115	0.002049	-0.008077	...	-0.000312	-0.008930	0.009262	0.011242	-0.01

5 rows × 100 columns