from keras.preprocessing import text from keras.utils import np\_utils from keras.preprocessing import sequence

**from** keras.preprocessing **import** text

**from** keras.utils **import** np\_utils

**from** keras.preprocessing **import** sequence

​

​

C:\Users\LENOVO\AppData\Roaming\Python\Python39\site-packages\scipy\\_\_init\_\_.py:146: UserWarning: A NumPy version >=1.17.3 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}"

**---------------------------------------------------------------------------**

**ImportError** Traceback (most recent call last)

Input **In [1]**, in <cell line: 2>**()**

1 **from** **keras.preprocessing** **import** text

**----> 2** **from** **keras.utils** **import** np\_utils

3 **from** **keras.preprocessing** **import** sequence

**ImportError**: cannot import name 'np\_utils' from 'keras.utils' (C:\Users\LENOVO\AppData\Roaming\Python\Python39\site-packages\keras\utils\\_\_init\_\_.py)

In [ ]:

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",

"where they have produced results comparable to and in some cases surpassing human expert performance"

]

*# dl\_data = data.split()*

In [ ]:

tokenizer **=** text.Tokenizer()

tokenizer.fit\_on\_texts(dl\_data)

word2id **=** tokenizer.word\_index

​

*# build vocabulary of unique words*

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** data]

​

vocab\_size **=** len(word2id)

embed\_size **=** 100

window\_size **=** 2 *# context window size*

​

print('Vocabulary Size:', vocab\_size)

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

In [ ]:

**from** keras.utils **import** pad\_sequences

**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)

In [ ]:

**import** numpy **as** np

i **=** 0

**for** x, y **in** generate\_context\_word\_pairs(corpus**=**wids, window\_size**=**window\_size, vocab\_size**=**vocab\_size):

*# print(x, y)*

**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

In [ ]:

**import** keras.backend **as** K

**from** keras.models **import** Sequential

**from** keras.layers **import** Dense, Embedding, Lambda

​

*# build CBOW architecture*

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')

​

*# view model summary*

print(cbow.summary())

In [ ]:

**for** epoch **in** range(1, 50):

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()

In [ ]:

**import** pandas **as** pd

weights **=** cbow.get\_weights()[0]

weights **=** weights[1:]

print(weights.shape)

​

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

In [ ]:

**from** sklearn.metrics.pairwise **import** euclidean\_distances

​

*# compute pairwise distance matrix*

distance\_matrix **=** euclidean\_distances(weights)

print(distance\_matrix.shape)

​

*# view contextually similar words*

similar\_words **=** {search\_term: [id2word[idx] **for** idx **in** distance\_matrix[word2id[search\_term]**-**1].argsort()[1:6]**+**1]

**for** search\_term **in** ['deep', 'unsupervised']}

​

similar\_words