untitled1

November 19, 2023

[1]:
```
pip install np_utils
```

```
Requirement already satisfied: np_utils in
c:\users\vaishnavi\anaconda3\lib\site-packages (0.6.0)
Requirement already satisfied: numpy>=1.0 in
c:\users\vaishnavi\anaconda3\lib\site-packages (from np_utils) (1.24.3)
Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
from keras.preprocessing import text
from keras.src.utils import np_utils
from keras.preprocessing import sequence
from keras.preprocessing.sequence import pad_sequences
import numpy as np
import pandas as pd
```

[3]:
```
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()
```

[4]:
```
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)]
```

[5]:
```
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=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
```

[6]:
```
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())
```

```
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
```

[7]:
```
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: 433.42123460769653

Epoch: 2        Loss: 429.14890909194946

Epoch: 3        Loss: 425.9578261375427

Epoch: 4        Loss: 422.91011905670166

Epoch: 5        Loss: 420.47326397895813
```

[8]:
```
weights = cbow.get_weights()[0]
weights = weights[1:]
print(weights.shape)
pd.DataFrame(weights, index=list(id2word.values())[1:]).head()
```

(74, 100)

[8]:
```
                0         1         2         3         4         5   \
deep     -0.053986 -0.032705 -0.054137  0.056982 -0.056818 -0.029501
networks -0.022894 -0.000098  0.005719 -0.055502 -0.002830 -0.011804
neural    0.021336 -0.046482 -0.040857  0.001032  0.017830 -0.023008
and       0.028876 -0.025039  0.048293 -0.021210 -0.006040 -0.018517
as       -0.047519 -0.025965  0.010029 -0.008076  0.042953 -0.043236

                6         7         8         9   …        90        91  \
deep     -0.024048 -0.010455  0.049077 -0.006933  …  0.006754 -0.039193
networks -0.038593 -0.004894 -0.043541  0.032048  …  0.049601  0.011592
neural   -0.017005  0.004414 -0.014917 -0.031787  …  0.030881 -0.010747
and      -0.022247 -0.012956  0.016154 -0.025678  … -0.048281 -0.022188
as       -0.048796  0.034132  0.017258  0.020821  …  0.005229  0.015517

                92        93        94        95        96        97  \
deep     -0.046910  0.043462  0.034635 -0.063149 -0.041819 -0.024874
networks -0.048512 -0.004966  0.041794  0.033884 -0.010356  0.043122
neural    0.022468  0.042298  0.037285  0.009902 -0.000136 -0.022752
and       0.028824 -0.034672  0.014509  0.006495 -0.021443  0.023626
as        0.014817  0.016218  0.022715 -0.046731  0.043029  0.009919

                98        99
deep     -0.007146  0.007089
networks  0.018750 -0.032835
neural    0.000485  0.032383
and      -0.042187 -0.024379
as        0.017243 -0.031168

[5 rows x 100 columns]
```

[9]:
```
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']
}

print(similar_words)
```

```
(74, 74)
{'deep': ['known', 'performance', 'results', 'analysis', 'artificial']}
```

[ ]: