In [1]:
```
file_path = r"C:\Users\Arman Sayyed\Desktop\DLL\CBOW(Ass5)\CBOW.txt"
with open(file_path, 'r') as file:
    file_content = file.read()
```

In [2]:
```
file_content
```

Out[2]:
```
'The speed of transmission is an important point of difference between the two viruses. Influenza has a shor
ter median incubation period (the time from infection to appearance of symptoms) and a shorter serial interv
al (the time between successive cases) than COVID-19 virus. The serial interval for COVID-19 virus is estima
ted to be 5-6 days, while for influenza virus, the serial interval is 3 days. This means that influenza can
spread faster than COVID-19. \n\nFurther, transmission in the first 3-5 days of illness, or potentially pre-
symptomatic transmission â€“transmission of the virus before the appearance of symptoms â€“ is a major drive
r of transmission for influenza. In contrast, while we are learning that there are people who can shed COVID
-19 virus 24-48 hours prior to symptom onset, at present, this does not appear to be a major driver of trans
mission. \n\nThe reproductive number â€“ the number of secondary infections generated from one infected indi
vidual â€“ is understood to be between 2 and 2.5 for COVID-19 virus, higher than for influenza. However, est
imates for both COVID-19 and influenza viruses are very context and time-specific, making direct comparisons
more difficult.  '
```

In [3]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.layers import Dense, GlobalAveragePooling1D, Embedding
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.metrics.pairwise import cosine_similarity
```

```
WARNING:tensorflow:From C:\Users\Arman Sayyed\AppData\Local\Programs\Python\Python39\lib\site-packages\keras
\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.
losses.sparse_softmax_cross_entropy instead.
```

In [19]:
```
sentences = file_content.split('.')

# Tokenize the sentences
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)
total_words = len(tokenizer.word_index) + 1

# Generate context-target pairs for training
window_size = 3
tokenized_sentences = tokenizer.texts_to_sequences(sentences)

data, labels = [], []
for sentence in tokenized_sentences:
    for i, target_word in enumerate(sentence):
        context = [
            sentence[j] for j in range(i - window_size, i + window_size + 1)
            if j != i and 0 <= j < len(sentence)
        ]
        data.append(context)
        labels.append(target_word)

# Convert data and labels to numpy arrays
data = pad_sequences(data)
labels = np.array(labels)
```

In [20]:
```
model = Sequential()
model.add(Embedding(input_dim = total_words, output_dim = 50, input_length = window_size*2))
model.add(GlobalAveragePooling1D())
model.add(Dense(total_words, activation = 'softmax'))

model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 6, 50)             5150

 global_average_pooling1d_2  (None, 50)                0
  (GlobalAveragePooling1D)

 dense_2 (Dense)             (None, 103)               5253

=================================================================
Total params: 10403 (40.64 KB)
Trainable params: 10403 (40.64 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [17]:
```

```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [17], in <cell line: 2>()
      1 data.shape
----> 2 data = data.reshape(198,6)
      3 data.shape

ValueError: cannot reshape array of size 6930 into shape (198,6)
```

In [21]:
```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(data, label, epochs=200, verbose=1)
```

```
7/7 [==============================] - 0s 3ms/step - loss: 3.5726 - accuracy: 0.2222
Epoch 49/200
7/7 [==============================] - 0s 3ms/step - loss: 3.5443 - accuracy: 0.2222
Epoch 50/200
7/7 [==============================] - 0s 3ms/step - loss: 3.5161 - accuracy: 0.2121
Epoch 51/200
7/7 [==============================] - 0s 6ms/step - loss: 3.4890 - accuracy: 0.2121
Epoch 52/200
7/7 [==============================] - 0s 3ms/step - loss: 3.4608 - accuracy: 0.2121
Epoch 53/200
7/7 [==============================] - 0s 3ms/step - loss: 3.4339 - accuracy: 0.2121
Epoch 54/200
7/7 [==============================] - 0s 3ms/step - loss: 3.4068 - accuracy: 0.2222
Epoch 55/200
7/7 [==============================] - 0s 3ms/step - loss: 3.3802 - accuracy: 0.2273
Epoch 56/200
7/7 [==============================] - 0s 3ms/step - loss: 3.3541 - accuracy: 0.2323
Epoch 57/200
7/7 [==============================] - 0s 3ms/step - loss: 3.3278 - accuracy: 0.2424
Epoch 58/200
```

In [23]:
```
word_embeddings = model.layers[0].get_weights()[0]
```

In [24]:
```
target_word = 'influenza'
target_embedding = word_embeddings[tokenizer.word_index[target_word]]

similarities = cosine_similarity(target_embedding.reshape(1, -1), word_embeddings)[0]

similar_index = similarities.argsort()[-5:][::1]

similar_words= [word for word, idx in tokenizer.word_index.items() if idx in similar_index]
print("Most Similar Word to Target word :", target_word, "is :", similar_words)
```

```
Most Similar Word to Target word : influenza is : ['influenza', 'means', 'spread', 'learning', 'higher']
```

In [ ]: