```
NAME = "Lilly Liu"
```

Lab 6: Skip Gram

Please read the following instructions very carefully

Working on the assignment / FAQs

- **Always use the seed/random_state as *42* wherever applicable** (This is to ensure repeatability in answers, across students and coding environments)
- The type of question and the points they carry are indicated in each question cell
- To avoid any ambiguity, each question also specifies what *value* must be set. Note that these are dummy values and not the answers
- If an autograded question has multiple answers (due to differences in handling NaNs, zeros etc.), all answers will be considered.
- You can delete the `raise NotImplementedError()`
- **Submitting the assignment** : Download the '.ipynb' file from Colab and upload it to bcourses. Do not delete any outputs from cells before submitting.
- That's about it. Happy coding!

Available software:

- Python's Gensim module: https://radimrehurek.com/gensim/ (install using pip)

Note: The most important hyper parameters of skip-gram/CBOW are vector size and windows size

```
!pip install gensim
import pandas as pd
import numpy as np
import gensim
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-packages (3.6.0)
Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.7/dist-packages (from gensim) (5.2
Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.7/dist-packages (from gensim) (1.15.0)
Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.7/dist-packages (from gensim) (1.21.6)
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.7/dist-packages (from gensim) (1.7.3)
```

```
import gensim.downloader as api

model = api.load('word2vec-google-news-300') # this step might take ~10-15 minutes
```

Q1 (1 point)

Find the cosine similarity between the following word pairs

- (France, England)
- (smaller, bigger)
- (England, London)
- (France, Rocket)
- (big, bigger)

```
#Replace 0 with the code / value; Do not delete this cell
similarity_pair1 = model.similarity('France', 'England')
similarity_pair2 = model.similarity('smaller', 'bigger')
similarity_pair3 = model.similarity('England', 'London')
similarity_pair4 = model.similarity('France', 'Rocket')
similarity_pair5 = model.similarity('big', 'bigger')
```

```
#This is an autograded cell, do not edit/delete
print(similarity_pair1, similarity_pair2, similarity_pair3, similarity_pair4, similarity_pair5)
```

```
0.39804944 0.7302272 0.43992856 0.07114174 0.68423855
```

Q2 (1 point)

Write an expression to extract the vector representations of the words:

- France
- England
- smaller
- bigger
- rocket
- big

Get only the first 5 elements for each vector representation.

```
#Replace 0 with the code / value to get the first 5 elements of each vector; Do not delete this cell
vector_1 = model['France'][:5]
vector_2 = model['England'][:5]
vector_3 = model['smaller'][:5]
vector_4 = model['bigger'][:5]
vector_5 = model['rocket'][:5]
vector_6 = model['big'][:5]


#This is an autograded cell, do not edit/delete
print(vector_1)
print(vector_2)
print(vector_3)
print(vector_4)
print(vector_5)
print(vector_6)
```

```
[0.04858398 0.07861328 0.32421875 0.03491211 0.07714844]
[-0.19824219  0.11523438  0.0625     -0.05834961  0.2265625 ]
[-0.05004883  0.03417969 -0.0703125   0.17578125  0.00689697]
[-0.06542969 -0.09521484 -0.06225586  0.16210938  0.01989746]
```

```
[-0.03198242  0.27148438 -0.2890625  -0.15429688  0.16894531]
[ 0.11132812  0.10595703 -0.07373047  0.18847656  0.07666016]
```

Q3 (1 point)

Find the euclidean distances between the word pairs :

- (France, England)
- (smaller, bigger)
- (England, London)
- (France, Rocket)
- (big, bigger)

```
#Replace 0 with the code / value; Do not delete this cell
eu_dist1 = np.linalg.norm(model['France']-model['England'])
eu_dist2 = np.linalg.norm(model['smaller']-model['bigger'])
eu_dist3 = np.linalg.norm(model['England']-model['London'])
eu_dist4 = np.linalg.norm(model['France']-model['Rocket'])
eu_dist5 = np.linalg.norm(model['big']-model['bigger'])
```

```
#This is an autograded cell, do not edit / delete
print(eu_dist1)
print(eu_dist2)
print(eu_dist3)
print(eu_dist4)
print(eu_dist5)
```

```
3.0151067
1.8618743
2.8752837
3.892071
1.9586496
```

Q4 (1 point)

Time to dabble with the power of Word2Vec. Find the 2 closest words for the following conditions:

- (King - Man + Queen)
- (bigger - big + small)
- (waiting - wait + run)
- (Texas + Milwaukee – Wisconsin)

Note: If your kernel crashes due to low memory and restarts, reload the model from the top and try running this part again.

```
#Replace 0 with the code / value; Do not delete this cell
closest1 = model.most_similar(positive=['King', 'Queen'], negative=['Man'])[:2]
closest2 = model.most_similar(positive=['bigger', 'small'], negative=['big'])[:2]
closest3 = model.most_similar(positive=['waiting', 'run'], negative=['wait'])[:2]
closest4 = model.most_similar(positive=['Texas', 'Milwaukee'], negative=['Wisconsin'])[:2]
```

```
#This is an autograded cell, do not edit/delete
print(closest1)
print(closest2)
print(closest3)
print(closest4)
```

```
[('Queen_Elizabeth', 0.5257916450500488), ('monarch', 0.5004087090492249)]
[('larger', 0.7402471899986267), ('smaller', 0.732999324798584)]
[('running', 0.5654535889625549), ('runs', 0.49640005826950073)]
[('Houston', 0.7767744064331055), ('Fort_Worth', 0.7270511388778687)]
```

Q5 (3 points)

Using the vectors for the words in the Google News dataset, apply K-means clustering (K=2) and find the top 5 most representative words/phrases of each cluster.

Note: Since there are ~3Mil words in the vocabulary, you can downsample it to 25k randomly selected words
Hint: The "similar_by_vector" method might be useful

Do not delete the below cell

```
# Replace 0 with the code / value; Do not delete this cell
# YOUR CODE HERE
from sklearn.cluster import KMeans
from random import sample

sampled = sample(list(model.wv.vocab), 25000)

samplev = {}
for s in sampled:
  samplev[s] = model.word_vec(s)
model_k = KMeans(n_clusters = 2, random_state = 42)
model_k = model_k.fit(list(samplev.values()))
centers = model_k.cluster_centers_
most_rep_cluster1 = np.array(model.similar_by_vector(centers[0], topn=5)).T[0]
most_rep_cluster2 = np.array(model.similar_by_vector(centers[1], topn=5)).T[0]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: DeprecationWarning: Call to deprecated `wv` (
```

```
#This is an autograded cell, do not edit/delete
print(most_rep_cluster1)
print(most_rep_cluster2)
```

```
['http_dol##.net_index###.html_http'
 'dol##.net_index####.html_http_dol##.net'
 'index###.html_http_dol##.net_index###.html' 'Deltagen_undertakes'
 'By_TRICIA_SCRUGGS']
['Emil_Protalinski_Published' 'By_QianMian_####-##-##'
 'By_HuDie_####-##-##' 'By_XiaoBing_####-##-##' 'BY_GEOFF_KOHL']
```

Q6 (1 point)

What loss function does the skipgram model use and briefly describe what this function is minimizing.

Do not delete the below cell

```
ans = "The skipgram model uses the categorical cross-entropy as the loss function. The function is minimizing the
```

Bonus Question (1 point)

Find at least 2 interesting word vec combinations like the ones given in Q4

Do not delete the below cell

```
# YOUR CODE HERE
print(model.most_similar(positive = ["college", "earnings"], negative = ["America"])[:2])
print(model.most_similar(positive = ["dog", "child"], negative = ["adult"])[:2])
```

```
[('executives_PGPX', 0.4177408218383789), ('EPS', 0.4163057506084442)]
[('puppy', 0.6720112562179565), ('pooch', 0.6400539875030518)]
```

✓ 0s completed at 1:28 PM