HW5-Week 8-Word2Vec Embeddings based on Emotions

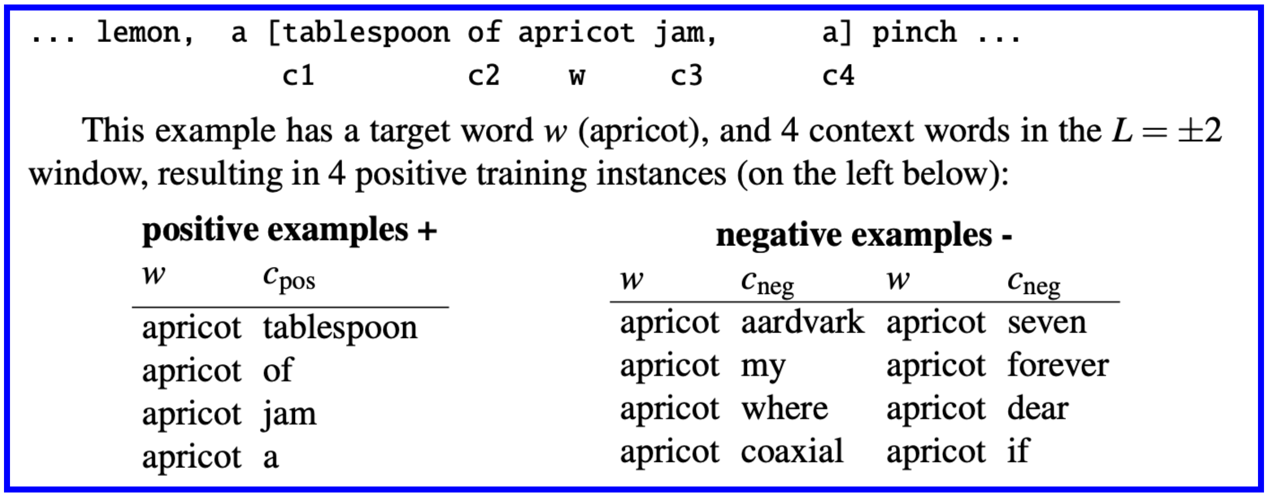
(100 points) **Discussion Week 8**
Due Sunday May 28th, 11:59 pm PST

Objective: In this assignment, you will investigate the core principles of the **Word2Vec skip-gram** algorithm.

Instructions: Complete all sections and submit the following:

1. A PDF or word doc answering the below questions.
2. Source code files **(e.g., Colab)** containing your implementation of the logistic regression classifier and data preprocessing steps.

Use the same data in the last homework with data of the 2 categories **`{Joy,Sadness}.Put all joy&sadness sentences into your corpus (there is no train/val/test split, use all rows as the training set).`** Remember the key of the implementation is that for each target word w, the neighboring words within the context window are positive words and we sample negative words. **Use context words in the L= +-2.**



```
... lemon,  a [tablespoon of apricot jam,     a] pinch ...
               c1          c2    w     c3       c4
```

This example has a target word *w* (apricot), and 4 context words in the $L = \pm 2$ window, resulting in 4 positive training instances (on the left below):

positive examples +

w	c_{pos}
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

w	c_{neg}	w	c_{neg}
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Colab file:
https://colab.research.google.com/drive/1JU0zjJXaHHrW3r0ujEdA6NxhgT0iOQbH?usp=sharing

- Make sure to upload the dataset as a csv file, because I used pandas read_csv function and also rename file to '**CS173-published-sheet.csv**'

Ellen Yim

Section 1: Construct Training Data (10 points)

1.1: **Tokenize the text:** Start by breaking documents with multiple sentences into individual sentences, each in its own row. Then tokenize each sentence into separate words, removing punctuation in the process. **Do not stem or lemmatize the words.**

- Code in colab [labeled under Section 1.1]

1.2: **Create a dictionary:** For all the unique words, create a dictionary that maps each word to a unique integer, and vice versa.

```
print('Joy dictionary: ')
for k, v in joy.items():
 print(k, v)

print()
print('Sadness dictionary: ')
for k, v in sadness.items():
 print(k, v)
```

- Run code before ‘Section 1.1’ to preprocess all data

1.3: **Construct the weight and context matrices** for learning. Use **d=5**. You can use the index of word to be the index of your matrix, for example, if aardvark in your dictionary has index 1, then it is corresponding to the first row in the **target embedding W and context embedding C**. **Initialize all the weights and embeddings to be one (every element in the vector is one).**

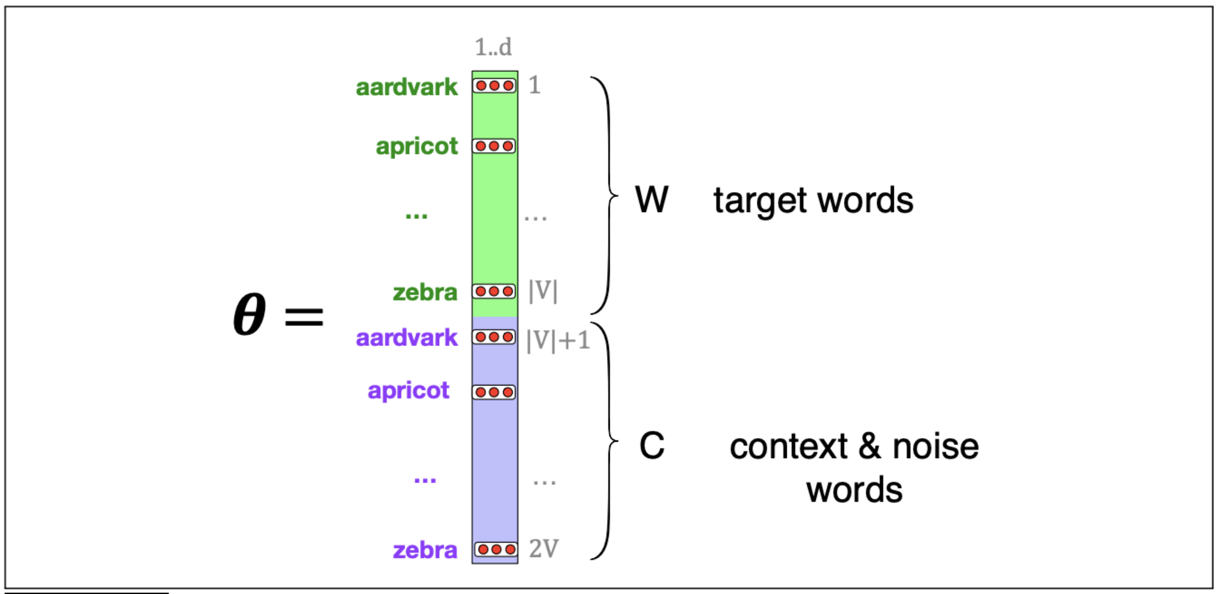


Figure 6.13 The embeddings learned by the skipgram model. The algorithm stores two embeddings for each word, the target embedding (sometimes called the input embedding) and the context embedding (sometimes called the output embedding). The parameter θ that the algorithm learns is thus a matrix of $2|V|$ vectors, each of dimension d, formed by concatenating two matrices, the target embeddings **W** and the context+noise embeddings **C**.

What is your |V|?

248

What is the dimension of your `W` and `C` matrices?
248 x 5

Section 2: Skip-gram with Negative Sampling (SGNS) (10 points)

For each positive word `c_pos` for the target word `w`, we'll create `k` negative samples, each consisting of the target `w` plus a 'noise word' `c_neg`. Implementing this sampling function following the textbook (**Section 6.8.2**):

- The noise words are chosen according to their weighted unigram frequency `p_α(w) in your training set`, where α is a weight. Use `α = 0.75.`

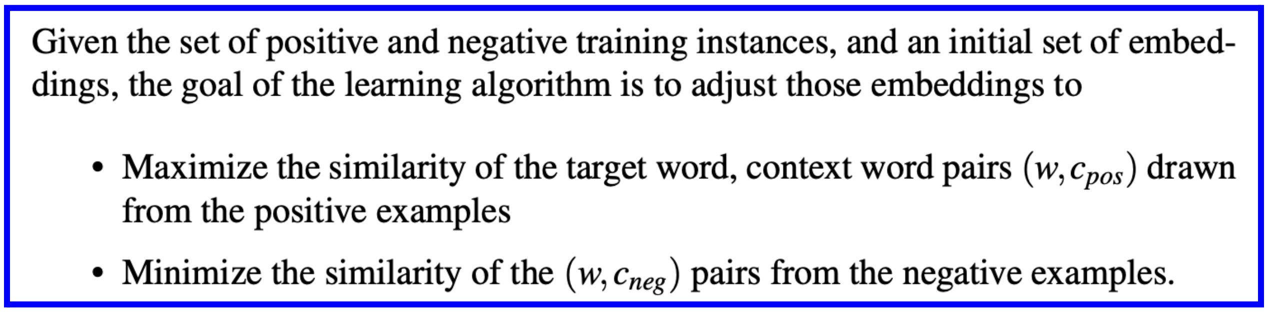
Give an example of your positive, target word pair and `k` negative pairs (k=5).

target word: laughter
positive word pairs of target word: ['the', 'and', 'of', 'children']
negative word pairs of target word: ['sunny', 'youth', 'zeal', 'proud', 'adorable']

- Code is in first cell under label ‘Section 2’

https://colab.research.google.com/drive/1JU0zjJXaHHrW3r0ujEdA6NxhgT0iOQbH#scrollTo=CcuWi78CYTNO

Section 3: Implementing Loss Computation (30 points)


Given the set of positive and negative training instances, and an initial set of embeddings, the goal of the learning algorithm is to adjust those embeddings to

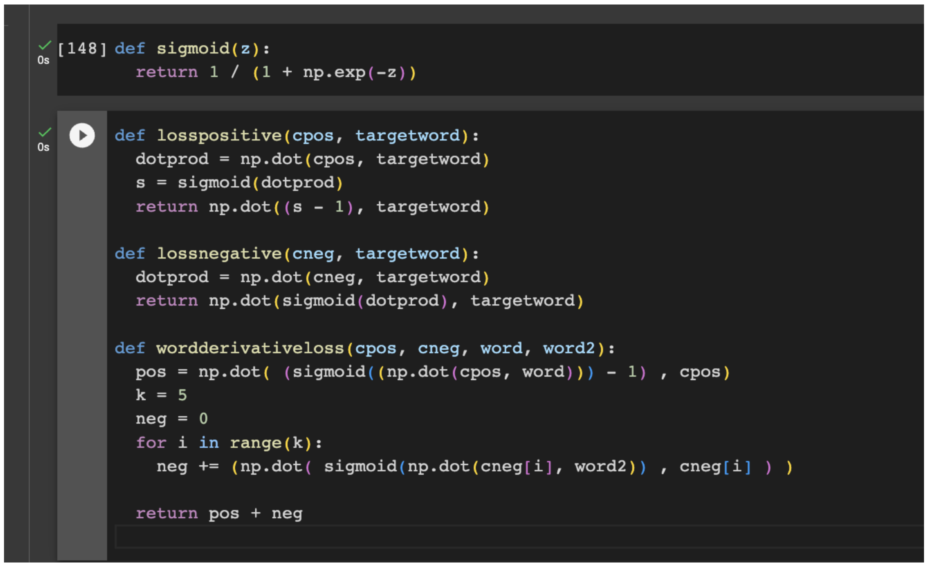
- Maximize the similarity of the target word, context word pairs (w, c_{pos}) drawn from the positive examples
- Minimize the similarity of the (w, c_{neg}) pairs from the negative examples.

Write Python code to compute the loss.

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{w} \quad (6.35)$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(\mathbf{c}_{neg} \cdot \mathbf{w})]\mathbf{w} \quad (6.36)$$

$$\frac{\partial L_{CE}}{\partial w} = [\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}) - 1]\mathbf{c}_{pos} + \sum_{i=1}^{k}[\sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w})]\mathbf{c}_{neg_i} \quad (6.37)$$



```
[148] def sigmoid(z):
        return 1 / (1 + np.exp(-z))
```

```
def losspositive(cpos, targetword):
  dotprod = np.dot(cpos, targetword)
  s = sigmoid(dotprod)
  return np.dot((s - 1), targetword)

def lossnegative(cneg, targetword):
  dotprod = np.dot(cneg, targetword)
  return np.dot(sigmoid(dotprod), targetword)

def wordderivativeloss(cpos, cneg, word, word2):
  pos = np.dot( (sigmoid((np.dot(cpos, word))) - 1) , cpos)
  k = 5
  neg = 0
  for i in range(k):
    neg += (np.dot( sigmoid(np.dot(cneg[i], word2)) , cneg[i] ) )

  return pos + neg
```

For the sample target word: ‘laughter’
Positive pairs loss: -0.12137019809031
Negative pairs loss: 0.48958527363835036

- Code is under the label ‘Section 3’

https://colab.research.google.com/drive/1JU0zjJXaHHrW3r0ujEdA6NxhgT0iOQbH#scrollTo=CsHZ_PjsYUso

Section 4: Learning & Optimization (30 points)

4.1: Complete the implementation for your SGD optimizer with different **learning rates = [0.00001, 0.0001, 0.001, 0.01, 0.1]** and fix the negative sampling k=5. What is your best learning rate and what is the lowest **training loss**? List all learning rate and loss here.

$$\mathbf{c}_{pos}^{t+1} = \mathbf{c}_{pos}^{t} - \eta[\sigma(\mathbf{c}_{pos}^{t} \cdot \mathbf{w}^{t}) - 1]\mathbf{w}^{t} \quad (6.38)$$

$$\mathbf{c}_{neg}^{t+1} = \mathbf{c}_{neg}^{t} - \eta[\sigma(\mathbf{c}_{neg}^{t} \cdot \mathbf{w}^{t})]\mathbf{w}^{t} \quad (6.39)$$

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta\left[[\sigma(\mathbf{c}_{pos} \cdot \mathbf{w}^{t}) - 1]\mathbf{c}_{pos} + \sum_{i=1}^{k}[\sigma(\mathbf{c}_{neg_i} \cdot \mathbf{w}^{t})]\mathbf{c}_{neg_i}\right] \quad (6.40)$$

Every word in your training corpus should be utilized as a target word. Therefore, your training algorithm should iterate over every token in each sentence, using the token and its neighboring words within the context window for learning.

- For my target words in the training corpus, I only did it for unique vocabulary.
- Below is a sample of my output, loss is using the ‘loss (pos/neg pairs) is using the loss functions created from previous section. Update is using the equations above (6.38, 6.39, 6.40)
- Code is labeled under ‘Section 4’ in Colab file.

- I am not too sure about the probabilities for (target, cpos) and (target, cneg) word pairs for the maximizing and minimizing similarity.

(Code snippet)

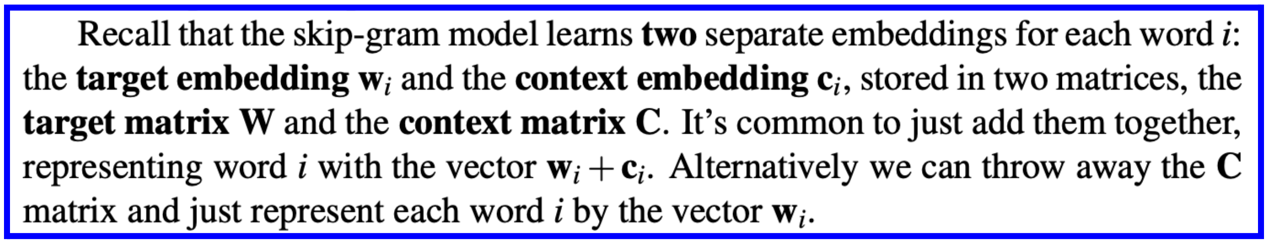
```
target= collapse
positive word pairs:  ['world', 'to', 'leaving', 'me']
negative word pairs:  ['laughter', 'rekindle', 'friendship', 'engaged', 'child']
loss (positive pairs):  -0.24028390695585256
loss (negative pairs):  0.7983023172894225
1e-05
update (positive):  [0.50063612 0.53669173 0.50047769 0.5098237 ]
update (negative):  [0.50030887 0.50015045 0.50030887 0.50015045 0.50110101]
0.0001
update (positive):  [0.50065774 0.53671335 0.50049931 0.50984533]
update (negative):  [0.50023703 0.5000786  0.50023703 0.5000786  0.50102917]
0.001
update (positive):  [0.500874   0.53692961 0.50071557 0.51006158]
update (negative):  [0.49951855 0.49936013 0.49951855 0.49936013 0.50031069]
0.01
update (positive):  [0.50303655 0.53909216 0.50287812 0.51222414]
update (negative):  [0.49233383 0.49217541 0.49233383 0.49217541 0.49312597]
0.1
update (positive):  [0.5246621  0.56071771 0.52450368 0.53384969]
update (negative):  [0.42048663 0.4203282  0.42048663 0.4203282  0.42127877]

target= hollow
positive word pairs:  ['in', 'that', 'part', 'of']
negative word pairs:  ['laughter', 'rekindle', 'friendship', 'engaged', 'child']
loss (positive pairs):  -0.2379195930302308
loss (negative pairs):  0.7983023172894225
1e-05
update (positive):  [0.51599819 0.51742245 0.50016081 0.53716439]
update (negative):  [0.50030887 0.50015045 0.50030887 0.50015045 0.50110101]
0.0001
update (positive):  [0.5160196  0.51744386 0.50018222 0.53718581]
update (negative):  [0.50023703 0.5000786  0.50023703 0.5000786  0.50102917]
0.001
update (positive):  [0.51623373 0.51765799 0.50039635 0.53739994]
update (negative):  [0.49951855 0.49936013 0.49951855 0.49936013 0.50031069]
0.01
update (positive):  [0.518375   0.51979927 0.50253762 0.53954121]
update (negative):  [0.49233383 0.49217541 0.49233383 0.49217541 0.49312597]
0.1
```

```
update (positive):  [0.53978777 0.54121203 0.52395039 0.56095397]
update (negative):  [0.42048663 0.4203282  0.42048663 0.4203282  0.42127877]
```

```
Best learning rate: 0.00001
Lowest training loss: 0.4562650529671598
```

Section 4: Final representation (10 points)

Add your W and C matrices for the final representation of all words in your vocab.

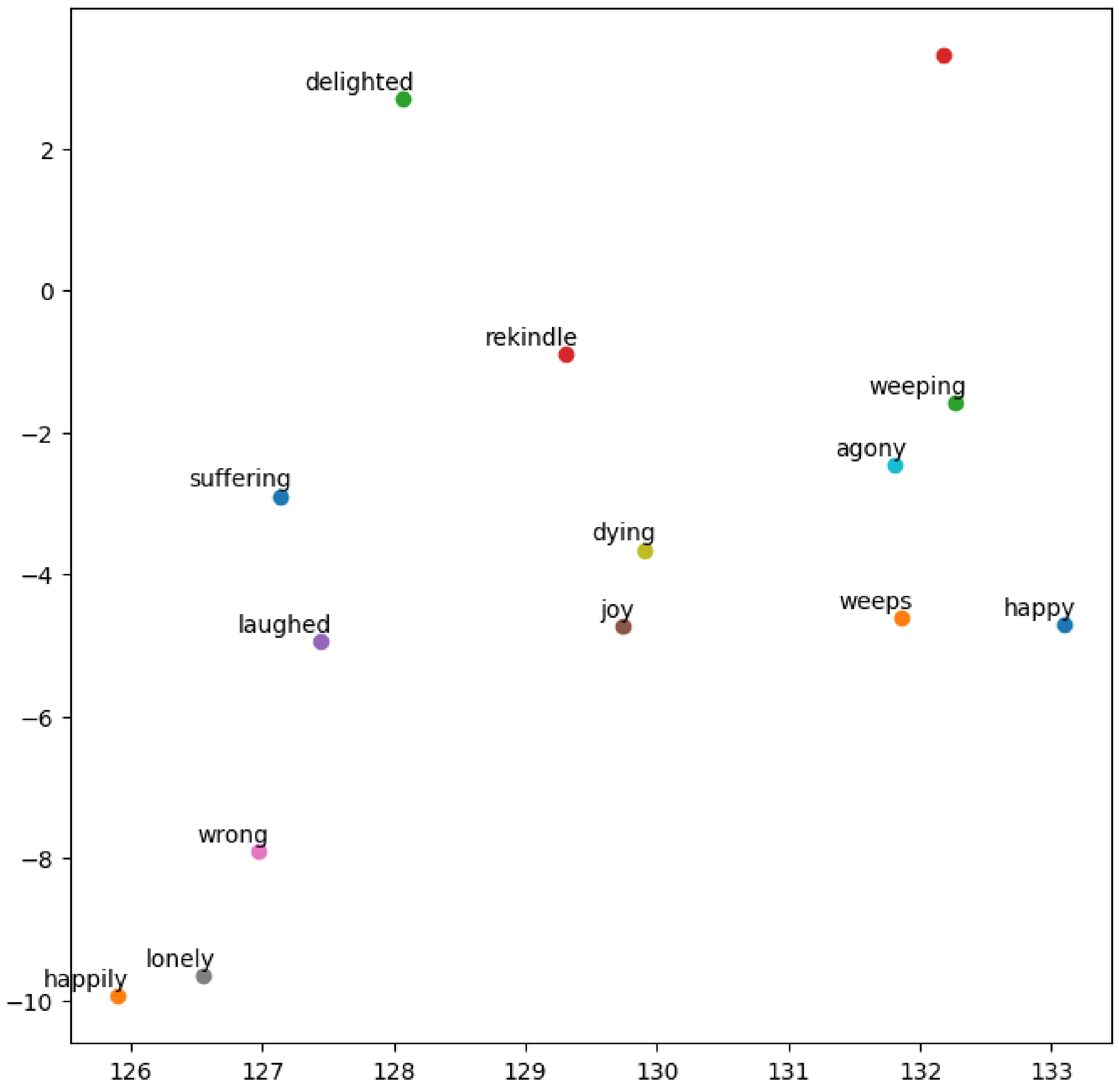


Recall that the skip-gram model learns **two** separate embeddings for each word i: the **target embedding** $\mathbf{w}_i$ and the **context embedding** $\mathbf{c}_i$, stored in two matrices, the **target matrix** $\mathbf{W}$ and the **context matrix** $\mathbf{C}$. It's common to just add them together, representing word i with the vector $\mathbf{w}_i + \mathbf{c}_i$. Alternatively we can throw away the $\mathbf{C}$ matrix and just represent each word i by the vector $\mathbf{w}_i$.

Using t-SNE (dimensionality reduction with t-distributed stochastic neighbor embedding) to reduce your word embeddings from 5 to 2. Draw the visualization for the following words:
`{happy, happily, delighted, rekindle, laughed, joy, wrong, lonely, dying, agony, suffering, weeps, weeping}`

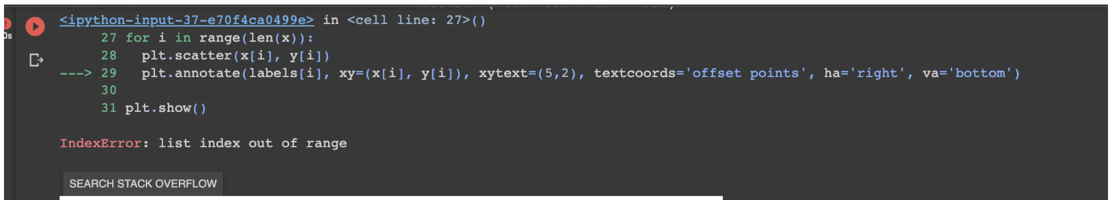
Copy and paste your plot here. You can find use existing package for the plot:

```
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
```

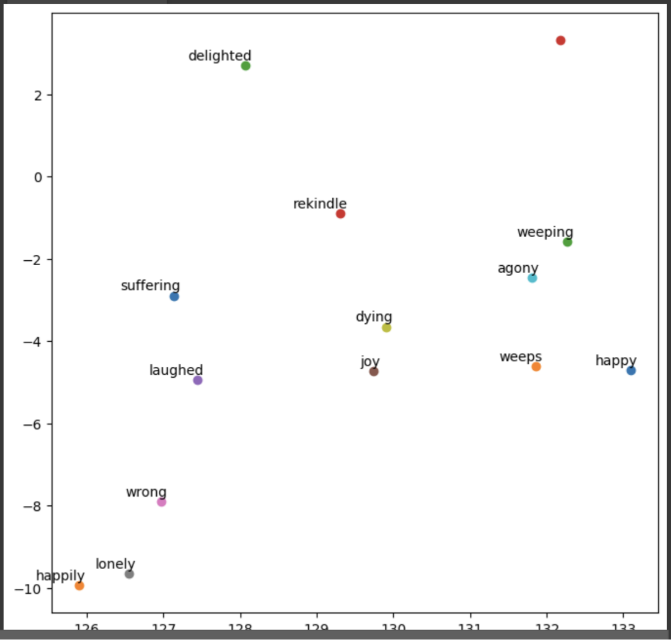


Code is under label 'Section 4: Final representation' in colab file.

- Code (in the very last cell) may take a while to run
- **When I run the cell, I keep getting a `list index out of range` error, but the visualization graph loads, as shown above. I'm not sure why I get that error. I referred to this when learning how to use the TSNE module**



```
<ipython-input-37-e70f4ca0499e> in <cell line: 27>()
     27 for i in range(len(x)):
     28   plt.scatter(x[i], y[i])
---> 29   plt.annotate(labels[i], xy=(x[i], y[i]), xytext=(5,2), textcoords='offset points', ha='right', va='bottom')
     30
     31 plt.show()

IndexError: list index out of range
```


delighted
rekindle
weeping
agony
suffering
dying
joy
weeps
happy
laughed
wrong
lonely
happily