Marin Szahic

- "I pledge my honor I've abided by the Stevens Honor System"

1. CODE

```python
# Normalize data
X = preprocessing.normalize(X, axis=0)

# Use OneHotEncoder to fully classify data
ohe = preprocessing.OneHotEncoder()
y = ohe.fit_transform(y).toarray()

# Split between training, testing and validation
X_train, X_rem, y_train, y_rem = train_test_split(X, y, test_size=0.2, random_state= 0, stratify=y)
X_valid, X_test, y_valid, y_test = train_test_split(X_rem, y_rem, test_size=0.5)

# Create Neural Net model with two dense layers
model = tf.keras.Sequential([
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])

# Compile model with optimizer and loss
model.compile(optimizer='adam',
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=['accuracy'])

# Fit Neural Network to data & print test accuarcy
model.fit(X_train, y_train, batch_size=10, epochs=150, validation_data = (X_valid, y_valid))

training_loss, training_acc = model.evaluate(X_train, y_train, verbose = 2)
validation_loss, validation_acc = model.evaluate(X_valid, y_valid, verbose = 2)
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)

print('\nTraining Accuracy: ', training_acc)
print('Validation Accuracy: ', validation_acc)
print('Test Accuracy: ', test_acc)
```

RESULTS

first 10 iterations

```
Epoch 1/150
12/12 [==============================] - 0s 12ms/step - loss: 1.1111 - accuracy: 0.3333 - val_loss: 1.1014 - val_accuracy: 0.4000
Epoch 2/150
12/12 [==============================] - 0s 2ms/step - loss: 1.1073 - accuracy: 0.3333 - val_loss: 1.1014 - val_accuracy: 0.4000
Epoch 3/150
12/12 [==============================] - 0s 2ms/step - loss: 1.1039 - accuracy: 0.3333 - val_loss: 1.0998 - val_accuracy: 0.4000
Epoch 4/150
12/12 [==============================] - 0s 2ms/step - loss: 1.1014 - accuracy: 0.3333 - val_loss: 1.0979 - val_accuracy: 0.4000
Epoch 5/150
12/12 [==============================] - 0s 2ms/step - loss: 1.0988 - accuracy: 0.3333 - val_loss: 1.0967 - val_accuracy: 0.4000
Epoch 6/150
12/12 [==============================] - 0s 2ms/step - loss: 1.0969 - accuracy: 0.3333 - val_loss: 1.0958 - val_accuracy: 0.4000
Epoch 7/150
12/12 [==============================] - 0s 2ms/step - loss: 1.0951 - accuracy: 0.3333 - val_loss: 1.0949 - val_accuracy: 0.4000
Epoch 8/150
12/12 [==============================] - 0s 2ms/step - loss: 1.0938 - accuracy: 0.3500 - val_loss: 1.0946 - val_accuracy: 0.4000
Epoch 9/150
12/12 [==============================] - 0s 2ms/step - loss: 1.0919 - accuracy: 0.6333 - val_loss: 1.0939 - val_accuracy: 0.6000
Epoch 10/150
12/12 [==============================] - 0s 2ms/step - loss: 1.0903 - accuracy: 0.6583 - val_loss: 1.0933 - val_accuracy: 0.6000
Epoch 11/150
```

Last 10 iterations

```
Epoch 140/150
12/12 [==============================] - 0s 2ms/step - loss: 0.4987 - accuracy: 0.9167 - val_loss: 0.4737 - val_accuracy: 0.9333
Epoch 141/150
12/12 [==============================] - 0s 2ms/step - loss: 0.4963 - accuracy: 0.9250 - val_loss: 0.4697 - val_accuracy: 0.9333
Epoch 142/150
12/12 [==============================] - 0s 2ms/step - loss: 0.4935 - accuracy: 0.9333 - val_loss: 0.4655 - val_accuracy: 0.9333
Epoch 143/150
12/12 [==============================] - 0s 2ms/step - loss: 0.4907 - accuracy: 0.9333 - val_loss: 0.4620 - val_accuracy: 0.9333
Epoch 144/150
12/12 [==============================] - 0s 2ms/step - loss: 0.4877 - accuracy: 0.9333 - val_loss: 0.4612 - val_accuracy: 0.9333
Epoch 145/150
12/12 [==============================] - 0s 2ms/step - loss: 0.4854 - accuracy: 0.9250 - val_loss: 0.4590 - val_accuracy: 0.9333
Epoch 146/150
12/12 [==============================] - 0s 2ms/step - loss: 0.4823 - accuracy: 0.9250 - val_loss: 0.4556 - val_accuracy: 0.9333
Epoch 147/150
12/12 [==============================] - 0s 2ms/step - loss: 0.4796 - accuracy: 0.9250 - val_loss: 0.4521 - val_accuracy: 0.9333
Epoch 148/150
12/12 [==============================] - 0s 2ms/step - loss: 0.4771 - accuracy: 0.9333 - val_loss: 0.4481 - val_accuracy: 0.9333
Epoch 149/150
12/12 [==============================] - 0s 2ms/step - loss: 0.4743 - accuracy: 0.9417 - val_loss: 0.4448 - val_accuracy: 0.9333
Epoch 150/150
12/12 [==============================] - 0s 2ms/step - loss: 0.4717 - accuracy: 0.9417 - val_loss: 0.4422 - val_accuracy: 0.9333
4/4 - 0s - loss: 0.4702 - accuracy: 0.9417 - 16ms/epoch - 4ms/step
1/1 - 0s - loss: 0.4422 - accuracy: 0.9333 - 13ms/epoch - 13ms/step
1/1 - 0s - loss: 0.4704 - accuracy: 0.9333 - 13ms/epoch - 13ms/step

Training Accuracy:  0.9416666626930237
Validation Accuracy:  0.9333333373069763
Test Accuracy:  0.9333333373069763
```

]- Accuracy

2) CODE

```python
import gensim, logging
from gensim.test.utils import datapath
from gensim.models import KeyedVectors
import numpy as np
from pyparsing imp (module) pyplot
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn.manifold import TSNE
from matplotlib.colors import Normalize
import pandas as pd
import time
import seaborn as sns

model = KeyedVectors.load_word2vec_format('vectors.txt', binary=False)

# most_similar is defaulted to cosine similarity
life_sim = model.most_similar(positive=["life"], topn= 20)
market_sim = model.most_similar(positive=["market"], topn= 20)
stanford_sim = model.most_similar(positive=["stanford"], topn= 20)

# Helper function to display similar words
def display_sim(arr, word):
    print("Most similar words for " + word)
    for i in range(20):
        print(f"{i}. {arr[i]}")
    print()

# Display Results
display_sim(life_sim, "life")
display_sim(market_sim, "market")
display_sim(stanford_sim, "stanford")

#===================   Part 2   =======================#

# Define function to gather cluster of words
def gather_cluster(word, model):
    sim_words = [word]
    cluster = [model[word]]
    neighbors = model.most_similar(positive=[word], topn=20)
    for neighbor in neighbors:
        sim_words.append(neighbor[0])
        cluster.append(model[neighbor[0]])
    return sim_words, [cluster]

# Plots 20 most similar words to given word
def tsne_plot_similar_words(title, label, embedding_cluster, word_cluster, a, filename=None):
```


```python
# Plots 20 most similar words to given word
def tsne_plot_similar_words(title, label, embedding_cluster, word_cluster, a, filename=None):
```

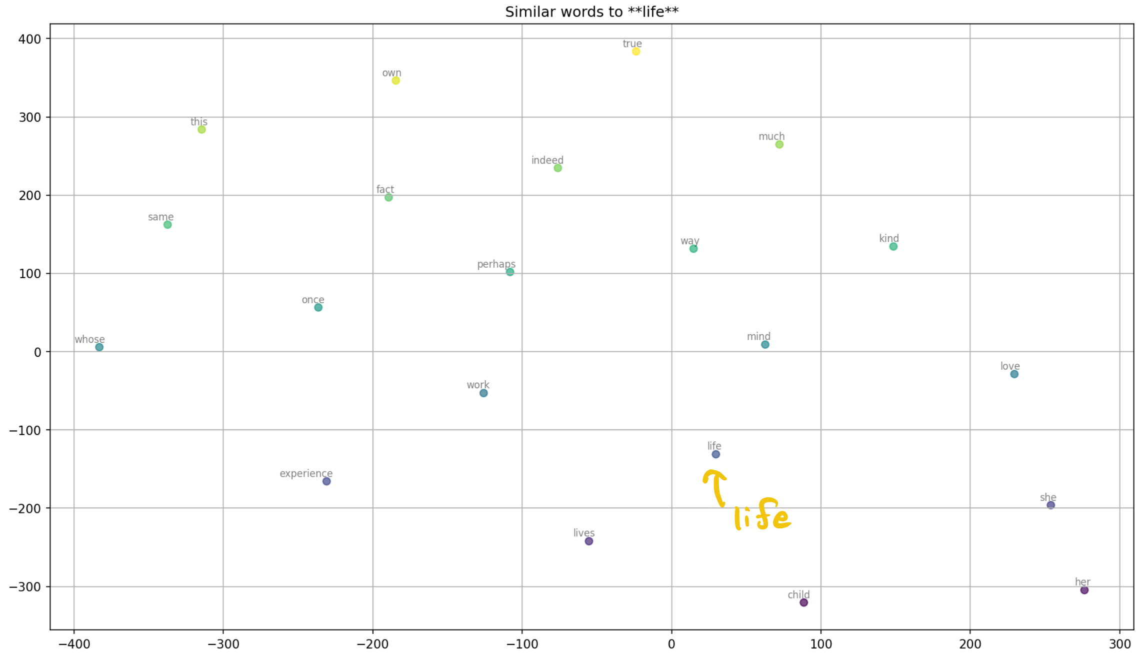

```python
    plt.figure(figsize=(16, 9))
    x = embedding_cluster[:, :, 0][0]
    y = embedding_cluster[:, :, 1][0]
    color = Normalize(min(y), max(y))
    plt.scatter(x, y, c=y, norm=color, alpha=a, label=label)
    for i, word in enumerate(word_cluster):
        plt.annotate(word, alpha=0.5, xy=(x[i], y[i]), xytext=(5, 2),
                     textcoords='offset points', ha='right', va='bottom', size=8)
    plt.title(title)
    plt.grid(True)
    if filename:
        plt.savefig(filename, format='png', dpi=150, bbox_inches='tight')
    plt.show()


# Plot for "life"
words, embeddings = gather_cluster("life", model)
embeddings = np.array(embeddings)
n, m, k = embeddings.shape

tsne_model_en_2d = TSNE(perplexity=15, n_components=2, init='pca', n_iter=3500, random_state=32)
embeddings_en_2d = np.array(tsne_model_en_2d.fit_transform(embeddings.reshape(n * m, k))).reshape(n, m, 2)

tsne_plot_similar_words('Similar words to **life**', "life", embeddings_en_2d, words, 0.7,
                        'similar_words_life.png')

# Plot for "market"
words, embeddings = gather_cluster("market", model)
embeddings = np.array(embeddings)
n, m, k = embeddings.shape


tsne_model_en_2d = TSNE(perplexity=15, n_components=2, init='pca', n_iter=3500, random_state=32)
embeddings_en_2d = np.array(tsne_model_en_2d.fit_transform(embeddings.reshape(n * m, k))).reshape(n, m, 2)

tsne_plot_similar_words('Similar words to **market**', "market", embeddings_en_2d, words, 0.7,
                        'similar_words_market.png')


# Plot for "stanford"
words, embeddings = gather_cluster("stanford", model)
embeddings = np.array(embeddings)
n, m, k = embeddings.shape
```

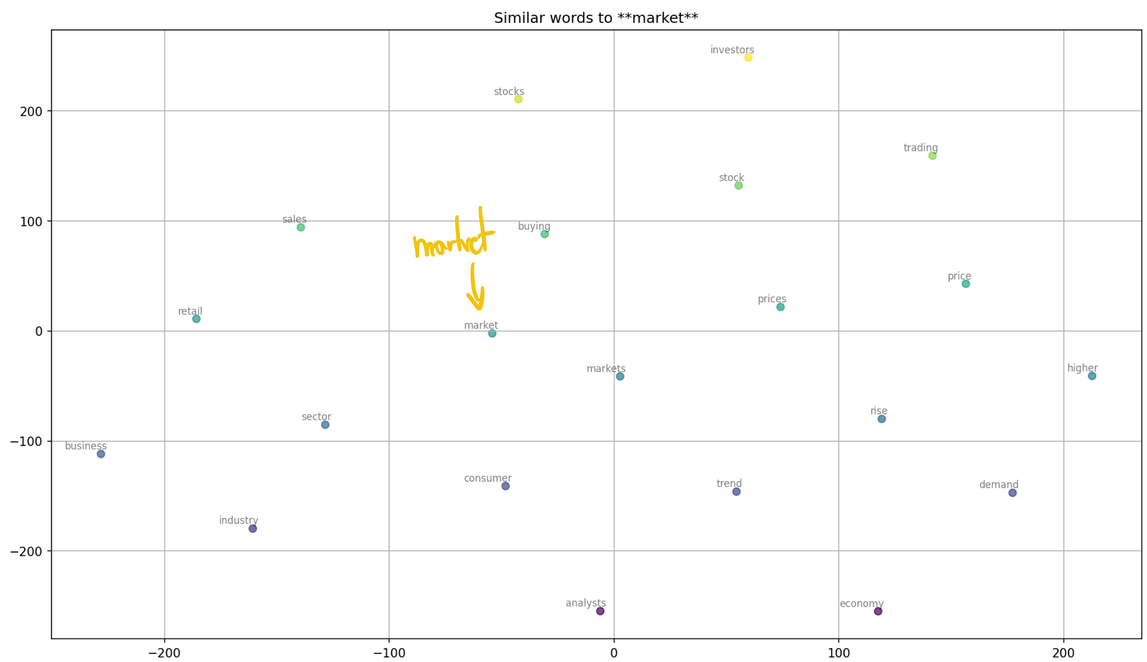
```python
tsne_model_en_2d = TSNE(perplexity=15, n_components=2, init='pca', n_iter=3500, random_state=32)
embeddings_en_2d = np.array(tsne_model_en_2d.fit_transform(embeddings.reshape(n * m, k))).reshape(n, m, 2)

tsne_plot_similar_words('Similar words to **stanford**', "stanford", embeddings_en_2d, words, 0.7,
                        'similar_words_stanford.png')

# Gather all words into np.array
words_wp = []
embeddings_wp = []

for word in list(model.vocab.keys()):
    embeddings_wp.append(model[word])
    words_wp.append(word)

embeddings_wp = np.array(embeddings_wp)
words_wp = np.array(words_wp)

# Convert dataset to dataframe
feat_cols = ['pixel' + str(i) for i in range(embeddings_wp.shape[1])]

df = pd.DataFrame(embeddings_wp, columns=feat_cols)
df['y'] = words_wp
df['label'] = df['y']

rndperm = np.random.permutation(df.shape[0])

# Take a subset of the dataframe to be plot
N = 100000
df_subset = df.loc[rndperm[:N],:].copy()
model_subset = df_subset[feat_cols].values

# Run TSNE timing each iteration
time_start = time.time()
tsne = TSNE(n_components=2, verbose=1, perplexity=40,n_iter=500)
tsne_results = tsne.fit_transform(model_subset)
print('t-SNE done! Time elapsed: {} seconds'.format(time.time()-time_start))

# Plot results
plt.figure(figsize=(16,10))
sns.scatterplot(
    x=tsne_results[:,0], y=tsne_results[:,1],
    data=df_subset,
    legend="full",
    alpha=0.1
)
```

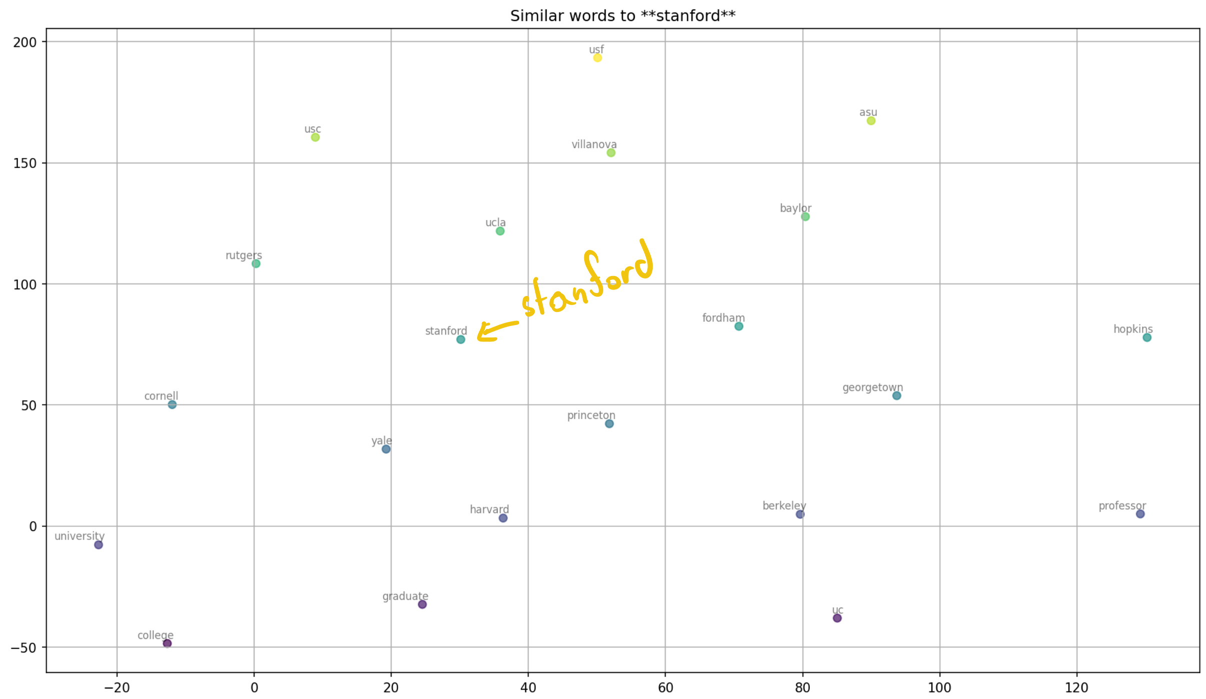
20 nearest words

```
Most similar words for life
0. ('mind', 0.8514840602874756)
1. ('love', 0.8403437733650208)
2. ('lives', 0.8392688632011414)
3. ('own', 0.836990475654602)
4. ('kind', 0.8338871598243713)
5. ('experience', 0.8213188648223877)
6. ('child', 0.8168196082115173)
7. ('perhaps', 0.8082367181777954)
8. ('she', 0.8081039786338806)
9. ('whose', 0.8071581721305847)
10. ('indeed', 0.8049614429473877)
11. ('her', 0.8037770390510559)
12. ('same', 0.8023735880851746)
13. ('work', 0.8022421598434448)
14. ('true', 0.8017044067382812)
15. ('way', 0.8002952933311462)
16. ('once', 0.8001490831375122)
17. ('fact', 0.7996558547019958)
18. ('this', 0.799416720867157)
19. ('much', 0.7988870143890381)

Most similar words for market
0. ('markets', 0.9401178956031799)
1. ('prices', 0.9033575654029846)
2. ('stock', 0.8850148916244507)
3. ('buying', 0.8556632995605469)
4. ('consumer', 0.8472995162010193)
5. ('retail', 0.8450345396995544)
6. ('stocks', 0.8400352001190186)
7. ('price', 0.8383353352546692)
8. ('sales', 0.8324212431907654)
9. ('business', 0.8298592567443848)
10. ('trend', 0.8276469707489014)
11. ('rise', 0.8247451186180115)
12. ('industry', 0.8199745416641235)
13. ('sector', 0.817125141620636)
14. ('investors', 0.8141677975654602)
15. ('trading', 0.8051414489746094)
16. ('demand', 0.8036580681800842)
17. ('economy', 0.802653968334198)
18. ('higher', 0.8007708191871643)
19. ('analysts', 0.7987768650054932)

Most similar words for stanford
0. ('ucla', 0.8524495959281921)
1. ('harvard', 0.846646249294281)
2. ('yale', 0.8393530249595642)
3. ('princeton', 0.834935188293457)
4. ('rutgers', 0.8128204345703125)
5. ('university', 0.7906179428100586)
6. ('baylor', 0.7722881436347961)
7. ('graduate', 0.7668696641921997)
8. ('georgetown', 0.7636426687240601)
9. ('cornell', 0.7606510519981384)
10. ('fordham', 0.7571069598197937)
11. ('asu', 0.7540745735168457)
12. ('usc', 0.7407923936843872)
13. ('uc', 0.7352784276008606)
14. ('hopkins', 0.7346756458282471)
15. ('usf', 0.7338851094245911)
16. ('professor', 0.7232831120491028)
17. ('berkeley', 0.722014725208282)
18. ('college', 0.7217254638671875)
19. ('villanova', 0.7199799418449402)
```

TSNE Plot of nearest words



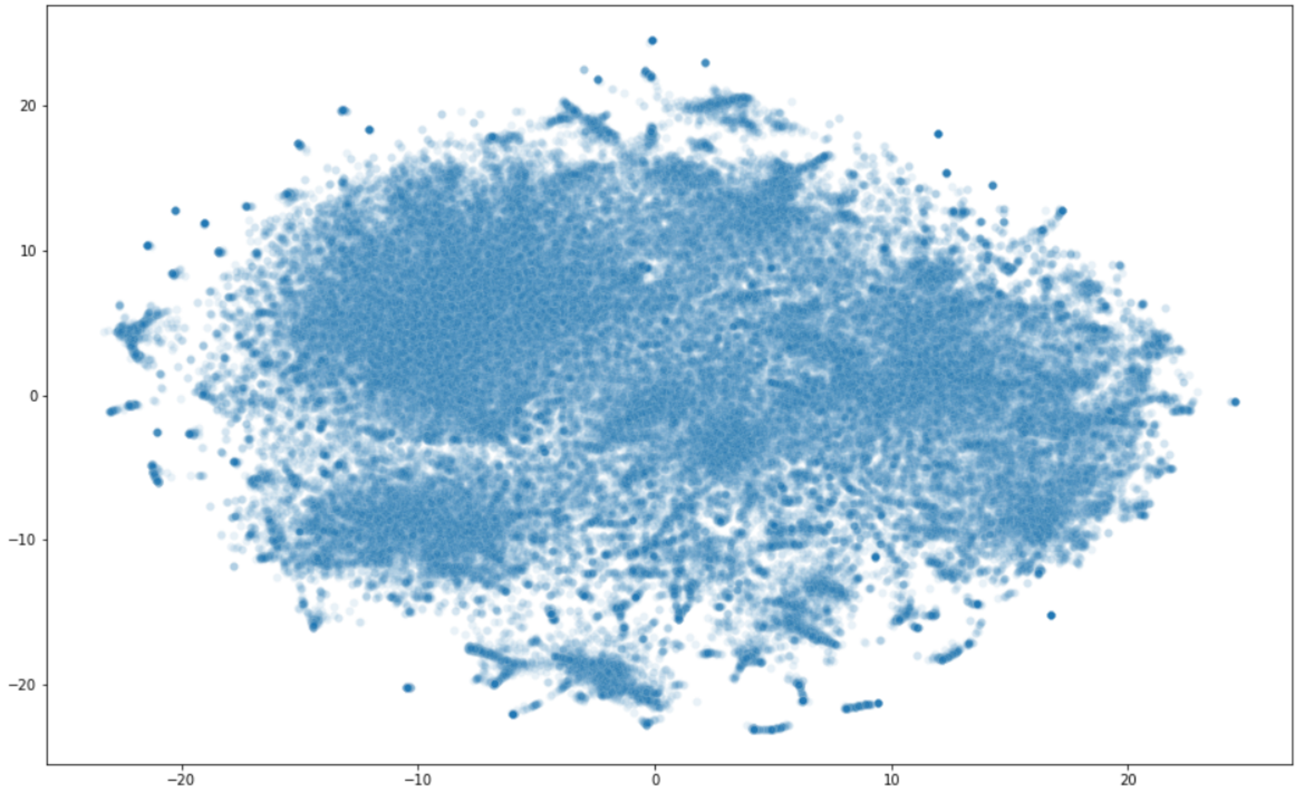




Plot TSNE of 100K words

(Random subset of 100k)

<matplotlib.axes._subplots.AxesSubplot at 0x7f980300c590>



Note : My computer couldn't handle 400k data points, & the cudatsne module suffers from several backwards compatability issues. So I took as large of a subset (100k) as I could. I hope that is acceptable.