In [1]:
```
## and you may ask yourself, "Well,how did I get here?"...

## TF WordEmbedding CBOW skip-gram Word2Vec 121220 and 070223
# word embedding, = vector-spaces where semantic-relationships between words modeled as distance between vecto
# CBOW, (Word2Vec probability-model #1) (minimize avg neg log-likelihood)(use context to predict current word)
# Skip-gram, (Word2Vec probability-model #2) (avg,neg,log) (use current word to predict neighbors (it's contex
# Word2vec (avg: gives mean error, neg: optimizer decreases error, log: products transformed to summation)
# Use tensorflow_hub and embed 2-3 words. Look at the resulting vectors.
```

In [2]:
```
# CBOW and Skip-gram are mirror-images.
# CBOW =  predict single word from fixed window size
# Skip-gram = predicts several context words from single input. Input = center word. Predictions = context wor
# Architecture: i)randomly-initialize word-embedding for word, ii)projection matrix NxD (num context-words X e
# ..generated each iteration, no hidden-layer, iii) vectors averaged together then fed into activation-functio
# ..index-probs in vector of dim V (size of vocab)
```

In [3]:
```
# 2-types of NLP: i) bag-of-words[Dense] (process sets or N-grams without order,) ii) sequence[Transformer] (w
# ** ratio between num-samples in train-data & mean num-of-words per sample determines whether to use i) or ii
# ** num_samples / mean_sample_length: if >1500 use Bag-of-bigrams, if <1500 use Sequence model
# ** example: 1000-word docs / 100,000 docs = ratio is 100, go with Bag-of-bigrams
# i) statistical language model to analyze text used for nlp based on word counts
# ii) sequence-to-sequence = encoder processes source sequence, decoder predicts future tokens in target seque
# neural-attention: creates context-aware word-reps = Transormer-architecture
# Transormer = TransformerEndcoder & TransformerDecoder
```

In [1]:
```
## START CBOW

# CBOW = continuous-bag-of-words which is a part of Word2Vec
# cbow model-architecture predicts current target-word based on source context-words
# we decide the vector-dims / embedding-dims, ex. embedding-dim = 2, then 'hello' = [0.213, 0.4543]
# cbow has input (1-hot), hidden (w + b), and output layers, then we pred via softmax
# workflow: prep data, create train-data, create cbow-model, train cbow-model, eval cbow-model
```

In [2]:
```
import gensim.downloader as api

info = api.info()  # show info about available models/datasets
model = api.load("glove-twitter-25")  # download the model and return as object ready for use
model.most_similar("cat")
```

```
[==================================================] 100.0% 104.8/104.8MB downloaded
```

Out[2]:
```
[('dog', 0.9590820074081421),
 ('monkey', 0.920357882976532),
 ('bear', 0.9143136739730835),
 ('pet', 0.9108031392097473),
 ('girl', 0.8880630731582642),
 ('horse', 0.8872726559638977),
 ('kitty', 0.8870542049407959),
 ('puppy', 0.886769711971283),
 ('hot', 0.886525571346283),
 ('lady', 0.884552001953125)]
```

In [3]:
```
from gensim.test.utils import common_texts
from gensim.models import Word2Vec

model=Word2Vec(sentences=common_texts, vector_size=100, window=5, min_count=1, workers=4)
model.save('word2vec.model')
```

In [4]:
```
model = Word2Vec.load('word2vec.model')
model.train([['hello', 'world']], total_examples=1, epochs=1)
```

Out[4]:
```
(0, 2)
```

In [5]:
```
vector=model.wv['computer']
sims=model.wv.most_similar('computer', topn=10)
```

In [6]:
```
from gensim.models import KeyedVectors
word_vectors = model.wv
word_vectors.save('word2vec.wordvectors')

wv=KeyedVectors.load('word2vec.wordvectors', mmap='r')
```

In [7]:
```
from gensim.test.utils import datapath
wv_from_text = KeyedVectors.load_word2vec_format(datapath('word2vec_pre_kv_c'), binary=False)
wv_from_bin = KeyedVectors.load_word2vec_format(datapath('euclidean_vectors.bin'), binary=True)
```

In [8]:
```
# end CBOW
```

In [ ]:
```

```

In [9]:
```
# Start Skip-Gram
```

In [10]:
```
import numpy as np
import pandas as pd
from nltk.tokenize.regexp import RegexpTokenizer
from nltk.corpus import stopwords
from gensim.models import Word2Vec, KeyedVectors  ## W2V shout out to Mr. Tomas M. 2010
from scipy.spatial.distance import cosine
```

In [11]:
```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Input, Dense
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style='darkgrid', context='talk')
```

In [12]:
```
# intentional ambiguity to try and confuse the model

text = ['Run jump run jump up and down outside.',
        'Old MacDonald had a jumper.',
        'The car is parked in the garage.',
        'The garage was built to house the cars.'
        'The house is attached to the garage which is good since its close by.',
        'The attached garage is good because its a short walk to the house.',
        'A short walk is enjoyable after dinner on the patio',
        'A patio is an enjoyable place for dinner.',
        'Dinner is after lunch.',
        'Lunch is before.',
        'Before lunch you can sit on the patio.',
        'The lion wanted breakfast and wandered near the patio.',
        'The wandering meanandering cheetah was near the lion.',
        'Old MacDonald ate a bag of cheetohs.',
        'Cheetohs are orange.',
        'Eating an orange on the patio while looking at the lion is fun said Old MacDonald.']
```

In [15]:
```
import nltk
nltk.download('stopwords')

def preprocess_text(document):
    tokenizer = RegexpTokenizer(r"[A-Za-z]{2,}")
    tokens = tokenizer.tokenize(document.lower())
    key_tokens = [token for token in tokens if token not in stopwords.words('english')]
    return key_tokens

corpus = []
for document in text:
    corpus.append(preprocess_text(document))
corpus
```

```
[nltk_data] Downloading package stopwords to C:\Users\Thank
[nltk_data]     you\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
```

Out[15]:
```
[['run', 'jump', 'run', 'jump', 'outside'],
 ['old', 'macdonald', 'jumper'],
 ['car', 'parked', 'garage'],
 ['garage',
  'built',
  'house',
  'cars',
  'house',
  'attached',
  'garage',
  'good',
  'since',
  'close'],
 ['attached', 'garage', 'good', 'short', 'walk', 'house'],
 ['short', 'walk', 'enjoyable', 'dinner', 'patio'],
 ['patio', 'enjoyable', 'place', 'dinner'],
 ['dinner', 'lunch'],
 ['lunch'],
 ['lunch', 'sit', 'patio'],
 ['lion', 'wanted', 'breakfast', 'wandered', 'near', 'patio'],
 ['wandering', 'meanandering', 'cheetah', 'near', 'lion'],
 ['old', 'macdonald', 'ate', 'bag', 'cheetohs'],
 ['cheetohs', 'orange'],
 ['eating',
  'orange',
  'patio',
  'looking',
  'lion',
  'fun',
  'said',
  'old',
  'macdonald']]
```

In [16]:
```
dimension = 2
window = 2
word2vec = Word2Vec(corpus, min_count=1, vector_size=dimension, window=window, sg=1)
word2vec.wv.get_vector('patio')
```

Out[16]:
```
array([-0.02654754,  0.0114735 ], dtype=float32)
```

In [17]:
```
n=3
word2vec.wv.most_similar(positive=['patio'], topn=n)
```

Out[17]:
```
[('looking', 0.9974758625030518),
 ('parked', 0.9954480528831482),
 ('cheetah', 0.9781259298324585)]
```

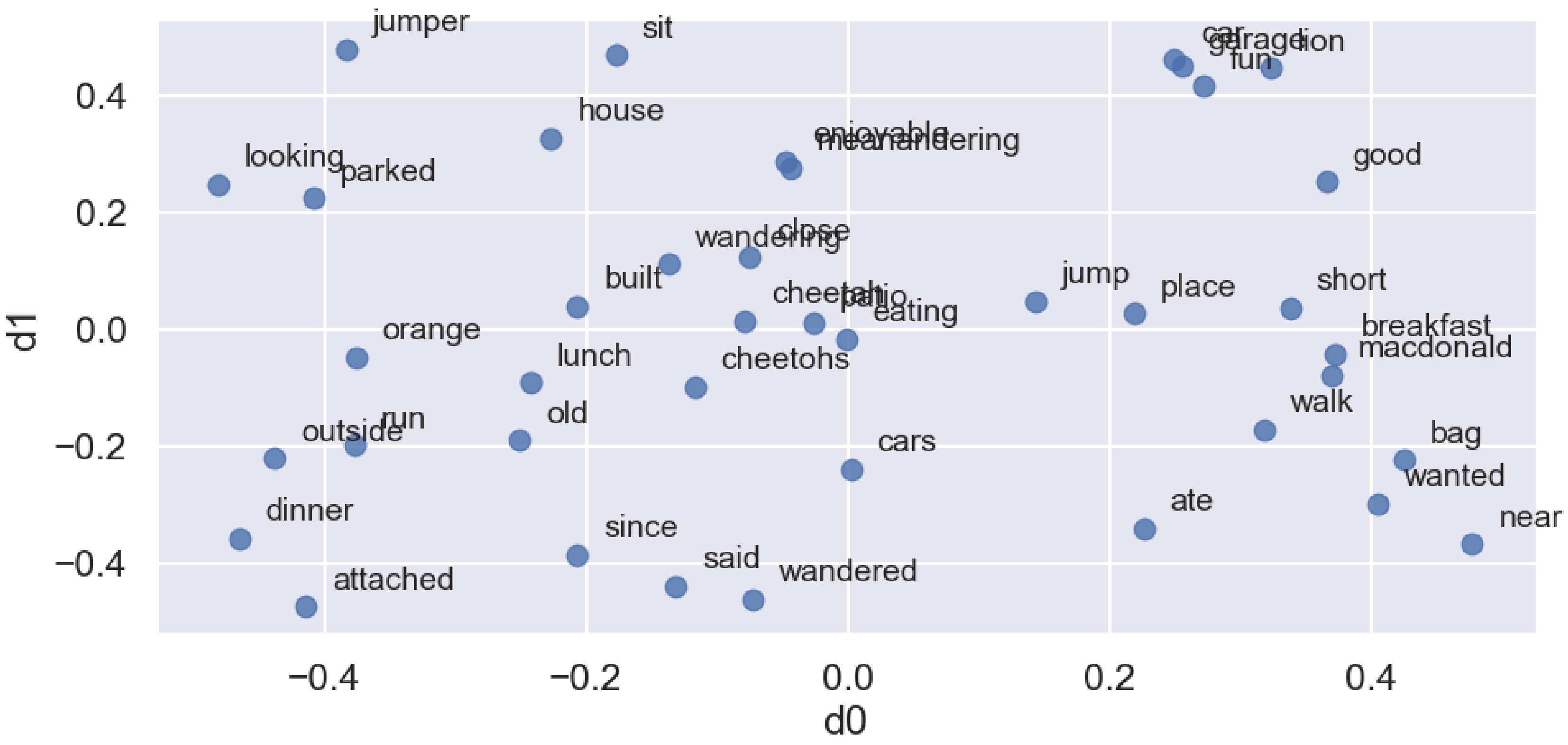
In [18]:
```
embedding = pd.DataFrame(columns=['d0', 'd1'])
for token in word2vec.wv.index_to_key:
    embedding.loc[token] = word2vec.wv.get_vector(token)
embedding
```

Out[18]:

	d0	d1
patio	-0.026548	0.011474
garage	0.254888	0.450171
dinner	-0.465057	-0.355795
lion	0.322857	0.448189
old	-0.250914	-0.188650
macdonald	0.369047	-0.076846
house	-0.226831	0.327703
lunch	-0.243008	-0.090801
jump	0.143574	0.049691
attached	-0.414787	-0.472955
good	0.365431	0.253374
short	0.337832	0.038106
walk	0.317540	-0.170268
enjoyable	-0.047320	0.288435
run	-0.376557	-0.196983
orange	-0.375989	-0.046632
near	0.476782	-0.365992
cheetohs	-0.117025	-0.097685
wanted	0.404098	-0.296785
cars	0.002258	-0.237687
looking	-0.480857	0.249733
outside	-0.438022	-0.219585
eating	-0.001667	-0.015021
jumper	-0.383062	0.480737
car	0.249021	0.461513
parked	-0.408057	0.224724
built	-0.206836	0.041369

	d0	d1
bag	0.424931	-0.223109
ate	0.225915	-0.339804
sit	-0.177424	0.469925
cheetah	-0.079127	0.015908
since	-0.207236	-0.384126
close	-0.075275	0.123525
meanandering	-0.044401	0.276683
wandering	-0.137284	0.112771
fun	0.271460	0.416540
wandered	-0.072549	-0.460507
place	0.218531	0.028655
breakfast	0.371399	-0.040853
said	-0.131668	-0.437716

In [19]:
```
sns.lmplot(data=embedding, x='d0', y='d1', fit_reg=False, aspect=2)
for token, vector in embedding.iterrows():
    plt.gca().text(vector['d0']+.02, vector['d1']+.03, str(token), size=14)
plt.tight_layout()
```



In [34]:
```
# End skip-gram
```

In [ ]:
```
```

In [20]:
```
# Start tb 2-gram & 3-gram
```

In [21]:
```
def split_Text_To_TWO_gram(string):

    words = string.split()

    TWO_gram_words = [' '.join(words[i: i + 2]) for i in range(0, len(words), 1)]

    return TWO_gram_words

split_Text_To_TWO_gram('this is an example sentence for the 2-gram words')
```

Out[21]:
```
['this is',
 'is an',
 'an example',
 'example sentence',
 'sentence for',
 'for the',
 'the 2-gram',
 '2-gram words',
 'words']
```

In [37]:
```
def split_Text_To_THREE_gram(string):

    words = string.split()

    THREE_gram_words = [' '.join(words[i: i + 3]) for i in range(0, len(words), 1)]

    return THREE_gram_words

split_Text_To_THREE_gram('this is an example sentence for the 3-gram words')
```

Out[37]:
```
['this is an',
 'is an example',
 'an example sentence',
 'example sentence for',
 'sentence for the',
 'for the 3-gram',
 'the 3-gram words',
 '3-gram words',
 'words']
```

In [38]:
```
# End tb 2-gram & 3-gram
```

In [ ]:
```
```

In [39]:
```
# Start embeddings & vectorization
```

In [40]:
```
import tensorflow as tf

path_to_file = tf.keras.utils.get_file('shakespeare.txt', 'https://storage.googleapis.com/download.tensorflow.
```

In [41]:
```
%%capture
with open(path_to_file) as f:
    lines = f.read().splitlines()
for line in lines[:20]:
    print(line)
```

In [42]:
```
text_ds = tf.data.TextLineDataset(path_to_file).filter(lambda x: tf.cast(tf.strings.length(x), bool))
```

In [43]:
```
def custom_standardization(splitTextToTriplet):
    lowercase = tf.strings.lower(splitTextToTriplet)
    return tf.strings.regex_replace(lowercase,
                                  '[%s]' % re.escape(string.punctuation), '')
```

In [44]:
```
vocab_size = 4096
sequence_length = 10
```

In [45]:
```
from tensorflow.keras import layers

vectorize_layer = layers.TextVectorization( standardize=custom_standardization,
                                            max_tokens=vocab_size,
                                            output_mode='int',
                                            output_sequence_length=sequence_length )
```

In [46]:
```
import re
re.compile('<title>(.*)</title>')
```

Out[46]:
```
re.compile(r'<title>(.*)</title>', re.UNICODE)
```

In [47]:
```
import string
```

In [48]:
```
vectorize_layer.adapt(text_ds.batch(1024))
```

In [49]:
```
inverse_vocab = vectorize_layer.get_vocabulary()
print(inverse_vocab[:18])
```

```
['', '[UNK]', 'the', 'and', 'to', 'i', 'of', 'you', 'my', 'a', 'that', 'in', 'is', 'not', 'for', 'with', 'm
e', 'it']
```

In [50]:
```
import tensorflow as tf
AUTOTUNE = tf.data.AUTOTUNE
```

In [51]:
```
text_vector_ds = text_ds.batch(1024).prefetch(AUTOTUNE).map(vectorize_layer).unbatch()
```

In [52]:
```
sequences = list(text_vector_ds.as_numpy_iterator())
print(len(sequences))
```

```
32777
```

In [53]:
```
for seq in sequences[:5]:
    print(f"{seq} => {[inverse_vocab[i] for i in seq]}")
```

```
[ 89 270   0   0   0   0   0   0   0   0] => ['first', 'citizen', '', '', '', '', '', '', '', '']
[138  36 982 144 673 125  16 106   0   0] => ['before', 'we', 'proceed', 'any', 'further', 'hear', 'me', 'sp
eak', '', '']
[34  0  0  0  0  0  0  0  0  0] => ['all', '', '', '', '', '', '', '', '', '']
[106 106   0   0   0   0   0   0   0   0] => ['speak', 'speak', '', '', '', '', '', '', '', '']
[ 89 270   0   0   0   0   0   0   0   0] => ['first', 'citizen', '', '', '', '', '', '', '', '']
```

In [54]:
```
## user-defined-functions for generate train data, W2V, and vizualizations on tensor-board can be added ##
```

In [70]:
```
## END 121220 ##
## END 070423 ##
```