In this exercise we use a dataset of movie lines to train Word Embeddings.

Prepare the data

Reading the text

Load the data (see .zip file in Learnweb) into the environment of this notebook.

If you are using google colab, you can upload the file as follows:

```
from google.colab import files

uploaded = files.upload()
```

Choose files movie_lines.tsv.zip

- **movie_lines.tsv.zip**(application/zip) - 8338070 bytes, last modified: 21/06/2022 - 100% done

```
Saving movie_lines.tsv.zip to movie_lines.tsv.zip
```

Extract the contents of the zip file.

```
from zipfile import ZipFile
with ZipFile('movie_lines.tsv.zip', 'r') as zipObj:
   zipObj.extractall()
```

Read the file and extract the text.

Note: reading the file with pandas or csv module does not work well, this file has a bad tsv format.

```
movie_lines = []
for line in open('movie_lines.tsv'):
    line = line.strip()
    while line[0] == '"' and line[-1] == '"':
        line = line[1:-1]
    movie_lines.append(line.split('\t')[-1])
movie_lines = [l.replace('""', '"').replace('  ', ' ') for l in movie_lines]
len(movie_lines)
```

```
304713
```

```
movie_lines[50:70]
```

```
["That's because it's such a nice one.",
 "I don't want to know how to say that though. I want to know useful things. I
 "Right. See? You're ready for the quiz.",
 "C'esc ma tete. This is my head",
 'Let me see what I can do.',
 'Gosh if only we could find Kat a boyfriend...',
 "That's a shame.",
 'Unsolved mystery. She used to be really popular when she started high school
 'Why?',
 'Seems like she could get a date easy enough...',
 "The thing is Cameron -- I'm at the mercy of a particularly hideous breed of
 'Cameron.',
 "No no it's my fault -- we didn't have a proper introduction ---",
 'Forget it.',
 "You're asking me out. That's so cute. What's your name again?",
 "Okay... then how 'bout we try out some French cuisine. Saturday? Night?",
 'Not the hacking and gagging and spitting part. Please.',
 "Well I thought we'd start with pronunciation if that's okay with you.",
 'Can we make this quick? Roxanne Korrine and Andrew Barrett are having an inc
 'I did.']
```

Pre-processing

Tokenization:

```
import nltk
nltk.download('punkt')
from nltk.tokenize import sent_tokenize
from nltk.tokenize import TweetTokenizer

tokenizer = TweetTokenizer(preserve_case=False)
```

```
    [nltk_data] Downloading package punkt to /root/nltk_data...
    [nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
sentences = []
for line in movie_lines:
    line_sents = sent_tokenize(line)
    for sent in line_sents:
        sentences.append(tokenizer.tokenize(sent))
len(sentences)
```

```
    514866
```

```
sentences[76:81]
```

```
    [['this', 'is', 'my', 'head'],
     ['let', 'me', 'see', 'what', 'i', 'can', 'do', '.'],
     ['gosh', 'if', 'only', 'we', 'could', 'find', 'kat', 'a', 'boyfriend', '...']
     ["that's", 'a', 'shame', '.'],
     ['unsolved', 'mystery', '.']]
```

Further processing, e.g. removing punctuation and stopwords?
While stopwords do not have lexical meaning, we still can follow meaning for other words from their distribution around stopwords (e.g. nouns often occur after determiners; being able to

distinguish words by their parts of speech can be important for semantic analyses). Thus, we should probably not remove those.
If you do, you have to justify your decision and should evaluate its effect.

```
# total number of words
len([c for clist in sentences for c in clist])

     3829713
```

```
# number of tokens (unique words)
from collections import Counter

words = Counter(c for clist in sentences for c in clist)
len(words)

     61245
```

```
str(words)[:154]

     'Counter({'.': 340514, 'you': 128297, '?': 110199, 'i': 103456, 'the': 99061,
     47335, 'and': 45812, '
```

Training Word Embeddings

```
# load library gensim (contains word2vec implementation)
import gensim

# ignore some warnings (probably caused by gensim version)
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.simplefilter(action='ignore', category=FutureWarning)

import multiprocessing
cores = multiprocessing.cpu_count()
```

```
gensim.__version__
# Note: this notebook was tested with gensim version 3.6.0

     '3.6.0'
```

Initializing the NN model

Parameters: (see
https://radimrehurek.com/gensim/models/word2vec.html#gensim.models.word2vec.Word2Vec
)

min_count = int - Ignores all words with total absolute frequency lower than this - (2, 100)
window = int - The maximum distance between the current and predicted word within a sentence. E.g. window words on the left and window words on the left of our target - (2, 10)
size = int - Dimensionality of the feature vectors. - (50, 300)
sample = float - The threshold for configuring which higher-frequency words are randomly downsampled. Highly influencial. - (0, 1e-5)
alpha = float - The initial learning rate - (0.01, 0.05)
min_alpha = float - Learning rate will linearly drop to min_alpha as training progresses. To set it: alpha - (min_alpha * epochs) ~ 0.00
negative = int - If > 0, negative sampling will be used, the int for negative specifies how many "noise words" should be drown. If set to 0, no negative sampling is used. - (5, 20)
workers = int - Use these many worker threads to train the model (=faster training with multicore machines)
sg = {0, 1}, optional - Training algorithm: 1 for skip-gram; otherwise CBOW.

```
import multiprocessing
cores = multiprocessing.cpu_count()

w2v_model = gensim.models.Word2Vec(min_count=5,
                                   window=3,
                                   size=100,  # this parameter is called 'vector_si
                                   workers=cores,
                                   sg=1
                                   )
```

Training the Word2Vec model with our data

```
# defining the vocabulary based on our data
w2v_model.build_vocab(sentences, progress_per=10000)

# training
w2v_model.train(sentences, total_examples=w2v_model.corpus_count, epochs=10, report
w2v_model.init_sims(replace=True)
```

Experiments with trained Word Embeddings

```
# word vectors are stored in model.wv
print("Size of the vocabulary: %d unique words have been considered" % len(w2v_mode

example_word = 'woman'
print("\nWord vector of " + example_word)
print(w2v_model.wv[example_word])
```

```
# nearest neighbours:
print("\nWords with most similar vector representations to " + example_word)
print(w2v_model.wv.most_similar(example_word))

# similarity between two word vectors:
print("\nCosine similarity to other words:")
print(w2v_model.similarity('woman', 'man'))
print(w2v_model.similarity('woman', 'tree'))
```

```
Size of the vocabulary: 17769 unique words have been considered

Word vector of woman
[-0.12428338 -0.154879    0.02684013  0.10706468  0.00602356  0.05294259
 -0.10254721 -0.08020414  0.02268961  0.08159889 -0.06778775  0.07279333
  0.20904592  0.02761954  0.16309908  0.01312446 -0.02207418 -0.17570199
 -0.08423298  0.10344184 -0.051408   -0.10489769  0.01280201  0.01921106
 -0.15175286 -0.10292951 -0.02342244  0.01513362 -0.03132872  0.10261379
 -0.00612064 -0.05966822  0.17442742 -0.05127969 -0.11301707  0.11288963
 -0.06731078 -0.06160271  0.12692006  0.20292267  0.02128376 -0.07685282
 -0.06652224 -0.06794152 -0.05727094 -0.04926075 -0.15791976  0.0044422
 -0.04881087  0.01922082 -0.03742855  0.13781556 -0.1734522  -0.10923117
 -0.10749488 -0.18500823 -0.05946709 -0.0079777   0.23452553  0.02346746
  0.07098631  0.07538388  0.02856915 -0.09444522  0.00726801 -0.04889245
  0.01656225  0.10075796 -0.05897082  0.09938398 -0.12204105 -0.08633889
 -0.10384272 -0.00558817  0.00399318 -0.05965995  0.13718161 -0.1292705
  0.09710084  0.19873092 -0.07535742  0.01253886 -0.04755069 -0.03744334
  0.02420999  0.20401229  0.17118718 -0.18249746 -0.07680031 -0.17946543
 -0.03748412 -0.00456074  0.15043983 -0.01023988  0.07289827 -0.06490348
 -0.06406686 -0.14558235 -0.05448155 -0.06962799]

Words with most similar vector representations to woman
[('girl', 0.8022258281707764), ('person', 0.7513905167579651), ('man', 0.71447

Cosine similarity to other words:
0.7144765
0.38344225
```

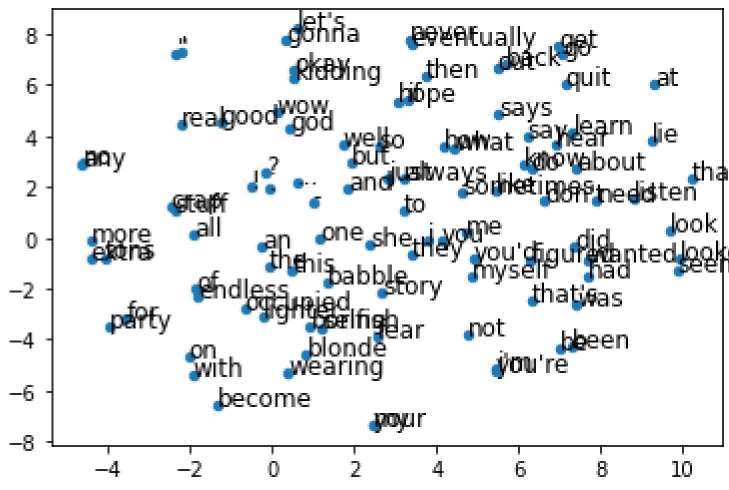
Note:
The calculated nearest neighbours (mostly) seem to make sense! Remember: we did not include any knowledge database into our model; all this meaning is extracted from the occurrences in the data based on the principle of Distributional Semantics.

Visualization of word vectors

We can display (some of) the vectors in a 2d-space by reducing the dimension with PCA and t-SNE.

```
import numpy as np
labels = []
count = 0
max_count = 100
X = np.zeros(shape=(max_count, len(w2v_model['woman'])))
```

```
for term in w2v_model.wv.vocab:
    X[count] = w2v_model[term]
    labels.append(term)
    count+= 1
    if count >= max_count: break

# It is recommended to use PCA first to reduce to ~50 dimensions
from sklearn.decomposition import PCA
pca = PCA(n_components=30)
X_50 = pca.fit_transform(X)

# Using t-SNE to further reduce to 2 dimensions
from sklearn.manifold import TSNE
model_tsne = TSNE(n_components=2, random_state=0)
Y = model_tsne.fit_transform(X_50)


# Show the scatter plot
import matplotlib.pyplot as plt
plt.scatter(Y[:,0], Y[:,1], 20)

plt.rcParams["figure.figsize"] = (20,10)

# Add labels
for label, x, y in zip(labels, Y[:, 0], Y[:, 1]):
    plt.annotate(label, xy = (x,y), xytext = (0, 0), textcoords = 'offset points',

plt.show()
```



We can also save the word vectors and look at (very cool) projections at https://projector.tensorflow.org/

```
# save words in words.tsv
# save corresponding word embedding vectors in vectors.tsv
with open("words.tsv", 'w') as words_outfile:
    with open("vectors.tsv", 'w') as vectors_outfile:
        words_outfile.write('word\n')
        for word in w2v_model.wv.vocab:
            words_outfile.write(word + '\n')
            vectors_outfile.write('\t'.join(str(value) for value in w2v_model.wv[wo
```

Analogy recovery

```
# we can use the parameters "positive" and "negative" to add/subtract vectors
w2v_model.wv.most_similar(positive=["woman"])
```

```
    [('girl', 0.8022258281707764),
     ('person', 0.7513905167579651),
     ('man', 0.7144765853881836),
     ('lady', 0.6926112174987793),
     ('actress', 0.6884387731552124),
     ('child', 0.6843668222427368),
     ('gal', 0.6526632308959961),
     ('prostitute', 0.6430126428604126),
     ('stenographer', 0.642153263092041),
     ('nun', 0.6420223712921143)]
```

"woman" is to "girl" like "man" is to ...?

"girl" - "woman" + "man"

```
w2v_model.wv.most_similar(positive=["girl", "man"], negative=["woman"])
```

```
    [('boy', 0.6911574602127075),
     ('guy', 0.6838798522949219),
     ('fella', 0.6712208986282349),
     ('gal', 0.6645423173904419),
     ('son-of-a-bitch', 0.6303056478500366),
     ('kid', 0.621398389339447),
     ('creep', 0.6083680391311646),
     ('cat', 0.6053136587142944),
     ('partner', 0.5994616746902466),
     ('snake', 0.5903835296630859)]
```

```
w2v_model.wv.most_similar(positive=["kitchen", "man"], negative=["woman"])
# note: these embeddings are trained on movie lines, which might include sexist bia
```

```
    [('garage', 0.6155099868774414),
     ('fridge', 0.6021116971969604),
     ('cellar', 0.5916866064071655),
     ('yard', 0.5876770615577698),
     ('cupboard', 0.586658239364624),
     ('deer', 0.5779769420623779),
     ('locker', 0.5775306820869446),
     ('shovel', 0.577021062374115),
     ('tide', 0.5743712186813354),
     ('grampa', 0.5729060769081116)]
```

```
w2v_model.wv.most_similar(positive=["pizza", "germany"], negative=["italy"])
# Not every attempt at analogy recovery returns results we would expect.
# This model is not perfect. The WEs are not based on that much data and the datase
```

```
[('tuna', 0.6161080598831177),
 ('sandwich', 0.614154577255249),
 ('crummy', 0.5961681604385376),
 ('whisky', 0.5896711349487305),
 ('collars', 0.5852710008621216),
 ('bono', 0.5842258930206299),
 ('curly', 0.5837256908416748),
 ('donut', 0.5834806561470032),
 ('low-life', 0.5828225016593933),
 ('freaky', 0.581141471862793)]
```

For further optimization of the training process you can use the parameter "negative" to use negative sampling instead of softmax. There is some discussion on that in the Word2Vec papers by Mikolov et al.

Exercises:

1. Set a reasonable value for the parameter "min_count" for training WEs on this dataset. You should always make such choices consciously and be able to explain your choice. Try to avoid "magic numbers". Give a (scientific) justification for your choice.
2. Perform a qualitative evaluation of the WE model.
    - Select (at least) 3 words and for each look at the nearest neighbours according to the model. Do the results make sense according to your idea of the meaning of the word?
    - Define (at least) 2 experiments for analogy recovery and look at the predictions for nearest neighbours according to the model. Do the results make sense?
    - Select an ambiguous word - a word which you expect to have been used in movie lines with different meanings. What is the similarity to a related word of one meaning? What is the similarity to a related word of another/the other meaning? (for example if you would select "mouse", you could compute the similarity to "rat" and to "keyboard" and would expect both values to be relatively high)
3. Remove stopwords and punctuation from the data before training Word Embeddings (you can use the lists below). Perform training and the evaluation above (2.) again. Do the results improve?

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

print(stopwords.words('english'))

stopwords_list = stopwords.words('english')
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
import string
string.punctuation
```

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

```
w2v_model_new= gensim.models.Word2Vec(min_count=6,
                                   window=3,
                                   size=100,  # this parameter is called 'vector_si
                                   workers=cores,
                                   sg=1
                                   )
# defining the vocabulary based on our data
w2v_model_new.build_vocab(sentences, progress_per=10000)

# training
w2v_model_new.train(sentences, total_examples=w2v_model.corpus_count, epochs=10, re
w2v_model_new.init_sims(replace=True)
```

```
# nearest neighbours:
print("\nWords with most similar vector representations to " + "woman")
print(w2v_model_new.wv.most_similar("woman"))


print("\nWords with most similar vector representations to " + "home")
print(w2v_model_new.wv.most_similar("home"))

print("\nWords with most similar vector representations to " + "car")
print(w2v_model_new.wv.most_similar("car"))
```

```
Words with most similar vector representations to woman
[('girl', 0.7751151323318481), ('person', 0.7368119955062866), ('man', 0.70823

Words with most similar vector representations to home
[('back', 0.6589435935020447), ('upstairs', 0.6564047336578369), ('visit', 0.6

Words with most similar vector representations to car
[('truck', 0.7768905162811279), ('cab', 0.6701630353927612), ('limo', 0.66870
```

```
print("\nFirst example for analogy recovery")
w2v_model_new.wv.most_similar(positive=["girl", "man"], negative=["woman"])
```

```
First example for analogy recovery
[('boy', 0.7061246633529663),
 ('guy', 0.6468604803085327),
 ('gal', 0.6431543231010437),
 ('kid', 0.6250653266906738),
 ('fella', 0.6244208812713623),
 ('creep', 0.587188184261322),
 ('son-of-a-bitch', 0.5866507291793823),
 ('cat', 0.5739247798919678),
 ('pal', 0.5711965560913086),
 ('brother', 0.5697212219238281)]
```

```
print("\nSecond example for analogy recovery")
w2v_model_new.wv.most_similar(positive=["rich", "queen"], negative=["king"])
```

```
Second example for analogy recovery
[('grown-up', 0.5328102707862854),
 ('good-looking', 0.5152576565742493),
 ('colored', 0.5127109885215759),
 ('struggling', 0.5122060775756836),
 ('famous', 0.5112366676330566),
 ('restless', 0.5070189237594604),
 ('horny', 0.5057792067527771),
 ('wealthy', 0.5042237639427185),
 ('weirdo', 0.4963825047016144),
 ('millionaire', 0.4942672550678253)]
```

```
# similarity between two word vectors:
print("\nCosine similarity for an ambigious word:")
print(w2v_model_new.similarity('orange', 'color'))
print(w2v_model_new.similarity('orange', 'fruit'))
```

```
Cosine similarity for an ambigious word:
0.33979797
0.57988155
```

I observe that for min_count of 6, though similarity values of some words have decreased, more similar words are being included. As we increase min_count, many similar words were disappearing as well as its values were decreasing.

Also with this min count also, the nearest neighbours of words considered makes sense. Analogy recovery also provides words which are in context

Here I considered the word orange which can mean the fruit as well as the color. I found that it has higher similarity to the fruit.

```
#Removing stopwords and punctuations
sentences_new=[]
for line in movie_lines:
    line_sents = sent_tokenize(line)
    for sent in line_sents:
        x = tokenizer.tokenize(sent)
        y = []
        for i in range(len(x)):

          if(x[i] not in stopwords_list):
            if(x[i].isalpha()): #Taking only alphabets and numbers
                y.append(x[i])
        sentences_new.append(y)
len(sentences_new)
```

```
514866
```

```
sentences_new[76:81]
```

```
[['head'],
 ['let', 'see'],
 ['gosh', 'could', 'find', 'kat', 'boyfriend'],
 ['shame'],
 ['unsolved', 'mystery']]
```

```
# total number of words
print(len([c for clist in sentences_new for c in clist]))
#here we see that the total number of words has increased
```

```
1428956
```

```
# number of tokens (unique words)
from collections import Counter

words = Counter(c for clist in sentences_new for c in clist)
print(len(words))
```

```
46913
```

```
#Evaluating question 2 again

w2v_model_new= gensim.models.Word2Vec(min_count=6,
                                  window=3,
                                  size=100,  # this parameter is called 'vector_si
                                  workers=cores,
                                  sg=1
                                  )
# defining the vocabulary based on our data
w2v_model_new.build_vocab(sentences_new, progress_per=10000)

# training
w2v_model_new.train(sentences_new, total_examples=w2v_model.corpus_count, epochs=10
w2v_model_new.init_sims(replace=True)

# nearest neighbours:
print("\nWords with most similar vector representations to " + "woman")
print(w2v_model_new.wv.most_similar("woman"))
print("\nWords with most similar vector representations to " + "home")
print(w2v_model_new.wv.most_similar("home"))

print("\nWords with most similar vector representations to " + "car")
print(w2v_model_new.wv.most_similar("car"))

print("\nFirst example for analogy recovery")
print(w2v_model_new.wv.most_similar(positive=["girl", "man"], negative=["woman"]))

print("\nSecond example for analogy recovery")
print(w2v_model_new.wv.most_similar(positive=["rich", "queen"], negative=["king"]))

# similarity between two word vectors:
print("\nCosine similarity for an ambigious word:")
print(w2v_model_new.similarity('orange', 'color'))
print(w2v_model_new.similarity('orange', 'fruit'))
```

```
Words with most similar vector representations to woman
[('man', 0.670150101184845), ('millionaire', 0.6200625896453857), ('girl', 0.6

Words with most similar vector representations to home
[('hawaii', 0.628600001335144), ('toto', 0.6274833679199219), ('pony', 0.62536

Words with most similar vector representations to car
[('truck', 0.7281874418258667), ('mulholland', 0.6513736248016357), ('bus', 0.

First example for analogy recovery
[('cy', 0.5374325513839722), ('guys', 0.5299738645553589), ('feller', 0.511576

Second example for analogy recovery
[('snuff', 0.5557448267936707), ('films', 0.5529403686523438), ('colored', 0.5

Cosine similarity for an ambigious word:
0.5815536
0.6778463
```

We can observe that words having more similar meaning were obtained earlier. When we remove stopwords, meanings maybe getting lost.

✓ 38s completed at 10:27 PM ● ×