ירדן רפאלי
203529482

NLP תרגיל 3

1) בשיטת GD עדכון:

$$w \leftarrow w - \alpha \nabla L = w - \alpha(\nabla L_0 - \lambda w)$$

ב- weight decay:

$$w \leftarrow (1-\lambda')w - \alpha \nabla L_0$$

נפרק את הביטוי הראשון:

$$w \leftarrow w - \alpha(\nabla L_0 - \lambda w) = w - \alpha \nabla L_0 + \alpha \lambda w = (1+\alpha\lambda)w - \alpha \nabla L_0$$

ונראה שהביטוי שווה ל- W.D כאשר:

$$1 - \lambda' = 1 + \alpha\lambda \iff \lambda' = -\alpha\lambda$$

3) נציג גרסה מותאמת של האלגוריתם Forward Backward:

נגדיר:

$$\forall i \in [N], \forall y, y' : M_i(y,y') = \exp(f(i,y,y',x)^T w)$$

באופן דומה, נגדיר טבלת עזר M' עם N עמודות ו-$|Y|$ שורות, כאשר הערך ב-$M'[i,y]$ הוא ההסתברות שהתג באינדקס i הוא y.

כעת נמלא את M' לפי איך שמגדירה האלגוריתם B.F פרט לשינוי הבא: נתיחס ל-y_3 כ-y_0 ונמצא באמצעות האלגוריתם את מחיר כל המסלולים החל מ-y_3.
כעת נוכל לכפול במכפלת שלושת התגים הראשונים:

```
for y_i ∈ Y:
    T[4, y_i] = M_4(y_3, y_i)

for k...N:
    for y_i ∈ Y:
        T[k, y_i] = Σ_{y_j ∈ Y} T[k-1, y_j] * M_k(y_j, y_i)

Sum ← Σ_{y_i ∈ Y} T[N, y_i] * M_N(y_i, STOP)

return Sum * Π_{i=1}^{3} M_i(y_{i-1}, y_i)
```

5) ראשית נמצא את הישר שעובר דרך x_1, y_1. כעת,

א. נעבור ישר חדש שעובר דרך x_2 ונפעל שיפוע זהה לישר שמצאנו (כלומר הישרים מקבילים). אם הישר החדש עובר דרך מילה אחרת נחזיר אותה בתור y_2 (גם יש כמה נבחר באקראי). אם הישר לא עובר, נשחק עם השיפוע שלו - נגדל לקדם אותו בצעד קטן כל פעם ולבדוק, או למצוא את הנק' שקרובות אליו.. ולהיעזר ב-cosine-similarity לחשב את השיפוע החדש.

ב. בעיה אפשרית היא שיכולים להיות כמה סוגים של relations לדוגמה פקיד עובד במלון, ופקיד ישן במלון (זה לא רחוק מהמציאות שעובדי המלון מתגוררים במלון וזה היה בקורפוס שלנו). ייתכן שצמד המילים "בנק" ו-"בספר" לא יימצאו במקרה השני לעיל. בנקים סגורים בלילה.

Question 4

In [14]:

```python
from collections import defaultdict
import pandas as pd
import numpy as np
from numpy.linalg import norm

np.set_printoptions(suppress=True)

stop_words = ['is', 'a', 'of', 'and']

def co_occurrence(sentences, window_size):
    histogram = defaultdict(int)
    vocab = set()
    for sentence in sentences:
        words = sentence.split(' ')
        words = list(filter(lambda x: x not in stop_words, words))
        for i in range(len(words)):
            word = words[i]
            vocab.add(word)
            rest_window = words[i + 1 : i + 1 + window_size]
            for neighbor_word in rest_window:
                key = tuple(sorted([neighbor_word, word]))
                histogram[key] += 1

    vocab = sorted(vocab)
    df = pd.DataFrame(data=np.zeros((len(vocab), len(vocab)), dtype=np.int16)
                      index=vocab,
                      columns=vocab)
    for key, value in histogram.items():
        df.at[key[0], key[1]] = value
        df.at[key[1], key[0]] = value
    return df
```

1) Co-occurence matrix.

In [15]:

```python
sentences = [
    'John likes NLP',
    'He likes Mary',
    'John likes machine learning',
    'Deep learning is a subfield of machine learning',
    'John wrote a post about NLP and got likes'
]

co_occurrence_df = co_occurrence(sentences, 1)
np.set_printoptions(linewidth=300)
print(co_occurrence_df)
```

```
          Deep  He  John  Mary  NLP  about  got  learning  likes  machine  \
Deep         0   0     0     0    0      0    0         1      0        0
He           0   0     0     0    0      0    0         0      1        0
John         0   0     0     0    0      0    0         0      2        0
Mary         0   0     0     0    0      0    0         0      1        0
NLP          0   0     0     0    0      1    1         0      1        0
about        0   0     0     0    1      0    0         0      0        0
got          0   0     0     0    1      0    0         0      1        0
learning     1   0     0     0    0      0    0         0      0        2
likes        0   1     2     1    1      0    1         0      0        1
machine      0   0     0     0    0      0    0         2      1        0
post         0   0     0     0    0      1    0         0      0        0
subfield     0   0     0     0    0      0    0         1      0        1
wrote        0   0     1     0    0      0    0         0      0        0
```

```
          post  subfield  wrote
Deep         0         0      0
He           0         0      0
John         0         0      1
Mary         0         0      0
NLP          0         0      0
about        1         0      0
got          0         0      0
learning     0         1      0
likes        0         0      0
machine      0         1      0
post         0         0      1
subfield     0         0      0
wrote        1         0      0
```

2) Singular Value Decomposition and eigenvalues.

In [16]:

```python
co_occurrence_matrix = co_occurrence_df.to_numpy()
u, s, v = np.linalg.svd(co_occurrence_matrix)
print(u, "\n\n", s, "\n\n", v, "\n\n")
print("eigenvalues:", s**2)
```

```
[[-0.09525618 -0.078084   -0.20409309 -0.30660432  0.10840231  0.06262278 -0.0
0986875  0.11843932 -0.22723703  0.01384834  0.69775862 -0.52978814  0.
]
 [-0.16408222 -0.20187228  0.11351611  0.08518163 -0.11456372 -0.16601357  0.0
282392  -0.14589432 -0.11169003 -0.22682727 -0.28939733 -0.44915337 -0.7071067
8]
 [-0.36361678 -0.45759106  0.28167501  0.26909965 -0.1014125  -0.07594124  0.4
5395368  0.19533118 -0.1045044   0.41073465  0.18797743  0.17719595  0.
]
 [-0.16408222 -0.20187228  0.11351611  0.08518163 -0.11456372 -0.16601357  0.0
282392  -0.14589432 -0.11169003 -0.22682727 -0.28939733 -0.44915337  0.7071067
8]
 [-0.25987227 -0.18258385  0.23872932  0.1509113   0.43876455  0.17183454 -0.5
2683758 -0.50615785 -0.14151891  0.12692599  0.11614814  0.11806938 -0.
]
 [-0.09294414  0.08314097  0.1288352  -0.15246158 -0.53761061  0.54461547 -0.2
9465061  0.09041336 -0.04576066  0.43260971 -0.19017074 -0.19639523  0.
]
 [-0.2390606  -0.14409564  0.2034698   0.01369787 -0.37933497 -0.04890169 -0.3
9549232  0.3097711   0.06910896 -0.56527546  0.27380553  0.28732963 -0.
]
 [-0.33015435  0.24675851 -0.54164539  0.64728074 -0.17963842  0.09188441 -0.0
122701  -0.13156362  0.17786789 -0.00519345  0.14389729 -0.08493307 -0.
]
 [-0.56870279  0.6379502   0.30126193 -0.17982925  0.18984878 -0.24358643  0.0
3511061  0.16206091  0.08742444  0.08506553 -0.0596818  -0.07200609  0.
]
 [-0.4167716  -0.37042327 -0.43305827 -0.4935293   0.05791577  0.00357016  0.0
0159817 -0.08250592  0.43636806  0.03863315 -0.19942192  0.12697215 -0.
]
 [-0.06226861 -0.08015554  0.10318815  0.17095449  0.45213457  0.6272624   0.1
6049     0.40572576  0.17733768 -0.28916475 -0.15536665 -0.14955451  0.
]
 [-0.21550319  0.03913235 -0.36727033 -0.07282907  0.0734532   0.06505598 -0.0
0858336  0.19271476 -0.7847237  -0.08916697 -0.26923915  0.26222778 -0.
]
 [-0.12287652  0.17016396  0.14501723 -0.20844511 -0.21164226  0.37574669  0.4
9419245 -0.54109773 -0.09304894 -0.32416617  0.1581298   0.17241938 -0.
]]

 [3.46596239 3.16016742 2.65391343 2.11112724 1.65714573 1.46726817 1.24332876
1.11081033 0.78274168 0.37502339 0.2062279  0.16031515 0.        ]

 [[-0.09525618 -0.16408222 -0.36361678 -0.16408222 -0.25987227 -0.09294414 -0.
2390606  -0.33015435 -0.56870279 -0.4167716  -0.06226861 -0.21550319 -0.122876
```

```
52]
 [ 0.078084    0.20187228  0.45759106  0.20187228  0.18258385 -0.08314097  0.1
4409564 -0.24675851 -0.6379502   0.37042327  0.08015554 -0.03913235 -0.1701639
6]
 [-0.20409309  0.11351611  0.28167501  0.11351611  0.23872932  0.1288352   0.2
034698  -0.54164539  0.30126193 -0.43305827  0.10318815 -0.36727033  0.1450172
3]
 [ 0.30660432 -0.08518163 -0.26909965 -0.08518163 -0.1509113   0.15246158 -0.0
1369787 -0.64728074  0.17982925  0.4935293  -0.17095449  0.07282907  0.2084451
1]
 [-0.10840231  0.11456372  0.1014125   0.11456372 -0.43876455  0.53761061  0.3
7933497  0.17963842 -0.18984878 -0.05791577 -0.45213457 -0.0734532   0.2116422
6]
 [ 0.06262278 -0.16601357 -0.07594124 -0.16601357  0.17183454  0.54461547 -0.0
4890169  0.09188441 -0.24358643  0.00357016  0.6272624   0.06505598  0.3757466
9]
 [-0.00986875  0.0282392   0.45395368  0.0282392  -0.52683758 -0.29465061 -0.3
9549232 -0.0122701   0.03511061  0.00159817  0.16049     -0.00858336  0.4941924
5]
 [-0.11843932  0.14589432 -0.19533118  0.14589432  0.50615785 -0.09041336 -0.3
097711   0.13156362 -0.16206091  0.08250592 -0.40572576 -0.19271476  0.5410977
3]
 [ 0.22723703  0.11169003  0.1045044   0.11169003  0.14151891  0.04576066 -0.0
6910896 -0.17786789 -0.08742444 -0.43636806 -0.17733768  0.7847237   0.0930489
4]
 [-0.01384834  0.22682727 -0.41073465  0.22682727 -0.12692599 -0.43260971  0.5
6527546  0.00519345 -0.08506553 -0.03863315  0.28916475  0.08916697  0.3241661
7]
 [ 0.69775862 -0.28939733  0.18797743 -0.28939733  0.11614814 -0.19017074  0.2
7380553  0.14389729 -0.0596818  -0.19942192 -0.15536665 -0.26923915  0.1581298
]
 [-0.52978814 -0.44915337  0.17719595 -0.44915337  0.11806938 -0.19639523  0.2
8732963 -0.08493307 -0.07200609  0.12697215 -0.14955451  0.26222778  0.1724193
8]
 [-0.         -0.70710678  0.          0.70710678  0.          0.          0.
-0.          -0.          0.          -0.          0.          0.         ]]


eigenvalues: [12.01289529  9.98665815  7.0432565   4.45685823  2.74613196  2.1
5287589  1.54586641  1.2338996   0.61268454  0.14064255  0.04252995  0.0257009
5  0.        ]
```

3) Reduced matrix.

In [17]:

```python
clipped_size = int(0.3 * s.shape[0])
u_tag = u[:, :clipped_size]
s_tag = np.diag(s[:clipped_size])
v_tag = v[:clipped_size, :]
x_tag = np.matmul(np.matmul(u_tag, s_tag), v_tag)
print(x_tag)
```

```
[[ 0.12272743 -0.05712672 -0.14543279 -0.05712672 -0.08856279 -0.01858134 -0.0
6683841  0.46327138  0.1820022   0.28075788 -0.0551122   0.27973583  0.0040097
1]
 [-0.05712672 -0.00127236 -0.00027247 -0.00127236  0.10323074  0.14471053  0.1
053263   0.1820022   0.82116207 -0.1297564   0.01536375  0.03687718  0.2221245
2]
 [-0.14543279 -0.00027247  0.00711724 -0.00027247  0.24194435  0.33367263  0.2
4501427  0.36801412  1.86444865 -0.33413496  0.03970333  0.05363304  0.5093334
5]
 [-0.05712672 -0.00127236 -0.00027247 -0.00127236  0.10323074  0.14471053  0.1
053263   0.1820022   0.82116207 -0.1297564   0.01536375  0.03687718  0.2221245
2]
 [-0.08856279  0.10323074  0.24194435  0.10323074  0.27996985  0.21331318  0.2
6109289  0.09658246  1.07119889 -0.11271466  0.07521302 -0.0160058   0.3007377
7]
 [-0.01858134  0.14471053  0.33367263  0.14471053  0.21331318  0.0521477   0.1
8444068 -0.14367498  0.11859449  0.08351377  0.07640111 -0.06643536  0.0444587
```

```
]
 [-0.06683841  0.1053263   0.24501427  0.1053263   0.26109289  0.18444068  0.2
4233526  0.09343942  0.92439281 -0.05719942  0.07081478 -0.00194264  0.2576073
9]
 [ 0.46327138  0.1820022   0.36801412  0.1820022   0.09658246 -0.14367498  0.0
9343942  0.96397902 -0.27976235  1.3882808  -0.01457163  0.74402926 -0.2005449
9]
 [ 0.1820022   0.82116207  1.86444865  0.82116207  1.07119889  0.11859449  0.9
2439281 -0.27976235  0.07571096  1.22204366  0.36683505  0.0522458   0.0150912
8]
 [ 0.28075788 -0.1297564  -0.33413496 -0.1297564  -0.11271466  0.08351377 -0.0
5719942  1.3882808   1.22204366  0.66612889 -0.12247618  0.77920931  0.2100226
1]
 [-0.0551122   0.01536375  0.03970333  0.01536375  0.07521302  0.07640111  0.0
7081478 -0.01457163  0.36683505 -0.12247618  0.02139338 -0.04415541  0.1093359
8]
 [ 0.27973583  0.03687718  0.05363304  0.03687718 -0.0160058  -0.06643536 -0.0
0194264  0.74402926  0.0522458   0.77920931 -0.04415541  0.51410537 -0.0706124
5]
 [ 0.00400971  0.22212452  0.50933345  0.22212452  0.30073777  0.0444587   0.2
5760739 -0.20054499  0.01509128  0.21002261  0.10933598 -0.07061245  0.0166380
1]]
```

One practical advantage is that we need much less numbers to express the co-occurence matrix (it's like JPEG compression in a way - we take x% of the crucial frequencies). The real advantage, however, is the reduced dimension, which means it's easier to work with our data (e.g. visualize, compute) and our data gets much more "smooth", it's continous rather than discrete.

4) Cosine similarity.

Now we're in the latent space (looking at U'), every word is described by only 3 features.

In [18]:

```
def cosine_similarity(a, b):
    return np.dot(a, b) / (norm(a) * norm(b))

john_vector = u_tag[2]
he_vector = u_tag[1]
subfield_vector = u_tag[11]
deep_vector = u_tag[0]
machine_vector = u_tag[9]

print("John-he:", cosine_similarity(john_vector, he_vector))
print("John-subfield:", cosine_similarity(john_vector, subfield_vector))
print("Deep-machine:", cosine_similarity(deep_vector, machine_vector))
```

```
John-he: 0.999241482928557
John-subfield: -0.15497560206457914
Deep-machine: 0.9329156098788491
```

As we can see, our toy model captures the semantic similarity to some extent. Since our dataset is so small, it might not make any sense, but we used the words `John` and `He` interchangeably and our model learned it! This is exciting.

On the other hand, our model knows that the words `John` and `Subfield` are not related because they are really far away from each other in our dataset - there are only few other words connecting them.

6) Cosine similarity - special case.

In [21]:

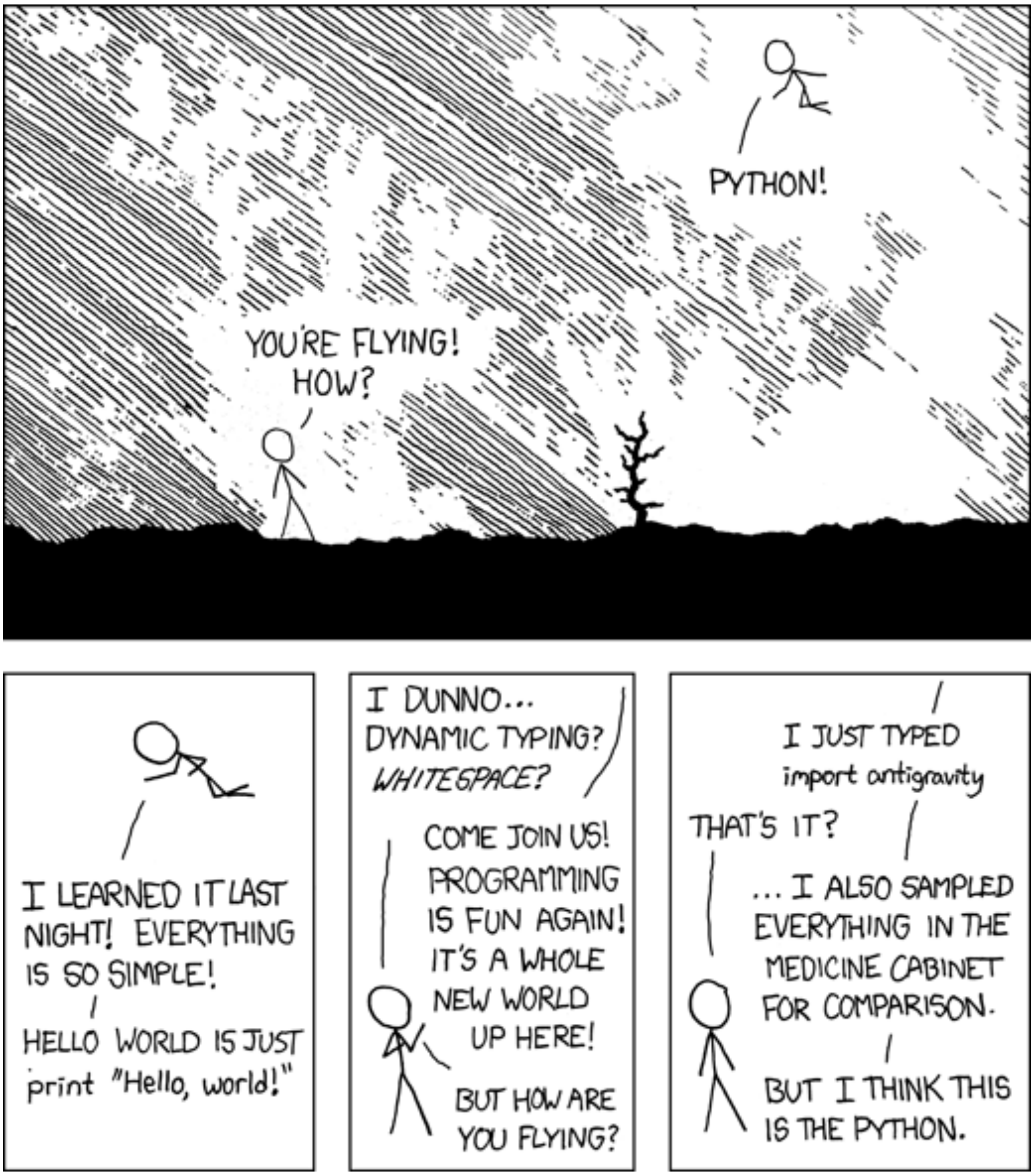
```
wrote_vector = u_tag[12]
post_vector = u_tag[10]
likes_vector = u_tag[8]
```

```
print("wrote-post:", cosine_similarity(wrote_vector, post_vector))
print("likes-likes:", cosine_similarity(likes_vector, likes_vector))
```

```
wrote-post: 0.243076522917071
likes-likes: 1.0
```

It might be a problem since these words are semantically-related. Maybe we can add more examples with these words so the smoothing introduced with the dimension reduction process won't butch it.

7) We would expect these two words to have a similarity of 1, because they are the same, but our model turns out to outsmart us - we used these 2 words with 2 different semantic meanings! So we're losing data here (it feels like quantization). Maybe, we can use a POS tagged corpus (where milenial `likes` is a noun), and define an entity also by it's tag. This way we will be able to differentiate between the two meanings.



Created with Jupyter using vscode. Not everything in 2020 sucks.