For next time….

We need to install a few more python libraries so that you can follow the demonstration in class.

You should install these three libraries using pip in your virtual environment (opening your working folder with vs code) or the global python environment (if you haven't been using virtual environments so far)

a. Numpy

Numpy is a math library that underlies many other python packages. We need to install a specific version for compatibility using the line:

```
pip install numpy==1.26.4
```

b. Pytorch

Pytorch is the most widely used machine learning / AI library.

You can install pytorch from here:

https://pytorch.org/get-started/locally/

select the correct options for your system then copy and paste the pip3 install line in your vs code terminal. For me it was something like:

```
pip3 install torch torchvision --index-url
https://download.pytorch.org/whl/cu126
```

If you are on Mac or on a windows machine without NVIDIA graphics card, then it will look different. If you do have an NVIDIA card, make sure to install the cuda enabled version as this will result in a massive performance gain when using AI models.

If you have an AMD card, you can install the ROCm version only on Linux or WSL in windows (only if you are comfortable with linux and know what you are doing)

c. Open CV

OpenCV is a computer vision and image processing library. You can install it with the following line:

```
pip install opencv-python==4.10.0.84
```

notice again the use of specific version

d. CLIP

CLIP is one of the most popular and important foundation models that allows us to embed (and therefore compare) images and text withing the same vector space. It underlies many image generation and analysis models.

The model can be found here:

https://github.com/openai/CLIP

in order to install we need a few more pieces of software to be present in our system.

Install GIT and GitHub clients from here:

https://git-scm.com/downloads

and

https://github.com/apps/desktop?ref_cta=download+desktop&ref_loc=installing+github+desktop&ref_page=docs

GIT is a versioning system that is used to manage and collaborate over code repositories.

Now in your VS code terminal you should be able to install CLIP using:

```
pip install git+https://github.com/openai/CLIP.git
```

e. After installation

After you install all the libraries you can open an empty python file and paste the following code:

```
import torch
import numpy as np
import cv2
import clip
from PIL import Image

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
version = torch.__version__
print(f"PyTorch version: {version}")

arr = np.array([1, 2, 3])
tensor = torch.from_numpy(arr).to(device)

print(tensor)
print(tensor.dtype)


model, preprocess = clip.load("ViT-B/32", device=device)

text = clip.tokenize(["a diagram", "a dog", "a cat"]).to(device)

with torch.no_grad():
    text_features = model.encode_text(text)
print(text_features)
```

If everything is installed correctly you shouldn't see any errors. Otherwise please come to office hours or post on Slack before next Monday.

Exercises

1. Semantic axis

Write a script that produces an output like below.

```
Axis: rural -> urban : building + (urban - rural)
          building:(0.82)
                 tower:(0.70)
                  buildings:(0.70)
                   design:(0.68)
                   skyscraper:(0.67)
                    towers:(0.65)
                     structure:(0.64)
                      construction:(0.63)
```

a. First load a model (e.g. 'glove-wiki-gigaword-100')

b. select a world that will form the starting central concept to be translated (e.g. 'building')

c. select two words to form an axis of translation (e.g. rural->urban)

d. get the vectors for the three words from the model

e. calculate the translation vector by subtracting the two axis word vectors (e.g. urban_vec – rural_vec)

f. create the translated vector by adding the translation to the central concept (e.g. building_vec + (urban_vec – rural_vec)

g. query the model using the similar_by_vector method for the 8 closest matches to the translated vector

h. print the results using the shown pattern.

i. bonus if you make the number of spaces (indentation) of each word to be related to its similarity value.

> Use your own words, give at least 3 examples with different word combinations. For each example try also to reverse the direction by changing the order of the subtracted words. (e.g. try both urban->rural and rural->urban)

2. Bi-axial semantics

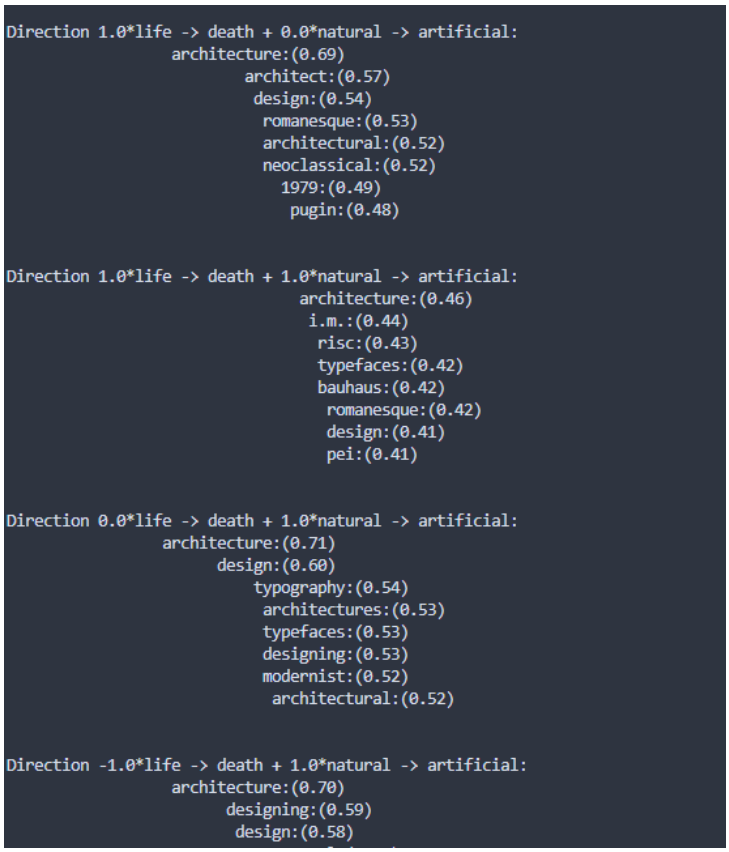
For this exercise, you need to start with a central concept C and two axes X1->X2 and Y1->Y2 (total 5 words)

Form the axis vectors using the model as in assignment 1:
C_vec

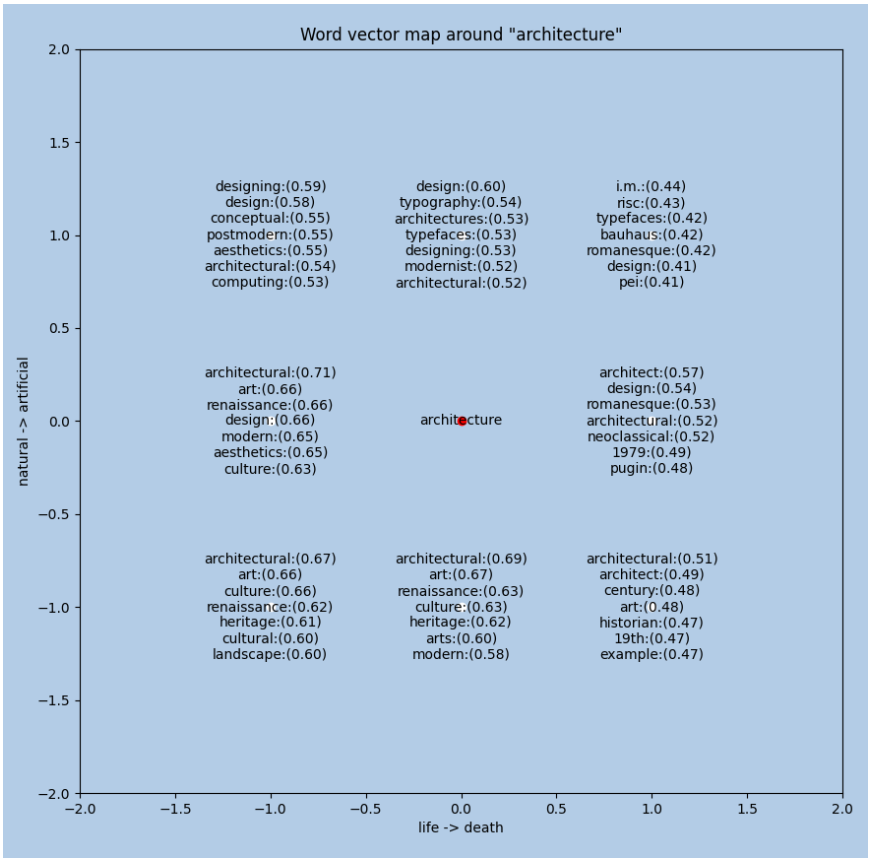
X_vec = X2_vec - X1_vec
Y_vec = Y2_vec – Y1_vec

Know repeat the assignment 1 by computing the following translations:
C + 1*X + 0*Y
C + 1*X + 1*Y
C + 0*X + 1*Y
C - 1*X + 1*Y
C - 1*X + 0*Y
C - 1*X - 1*Y
C + 0*X - 1*Y

for example, for C=’architecture’ with the axes {life->death} and {natural->artificial} we get something like the following:



These data conceptually allow us to explore a planar territory around a concept like in the following diagram:



Use the data that you calculated to create a diagram of the results. You can use any software or method you want for the graphical elements. The above example uses matplotlib directly in python, but you don't have to. You can use CAD, illustrator, hand-drawn or python. Try to give a character to the diagram that relates to the concepts that you used and the relative strength of the similarities (for example different color, opacity or font size depending on the similarity value, or slightly different placement)

3.Comparative (amended)

This exercise is based on the file "000_c_biaxial_dual_word_map.py" but with a small modification and using two models trained by you.

Train two gensim models on a two text files of your choice. These will become the model_a and model_b in the script

Now modify 000_c_biaxial_dual_word_map.py to use your models and relevant words of your choice.

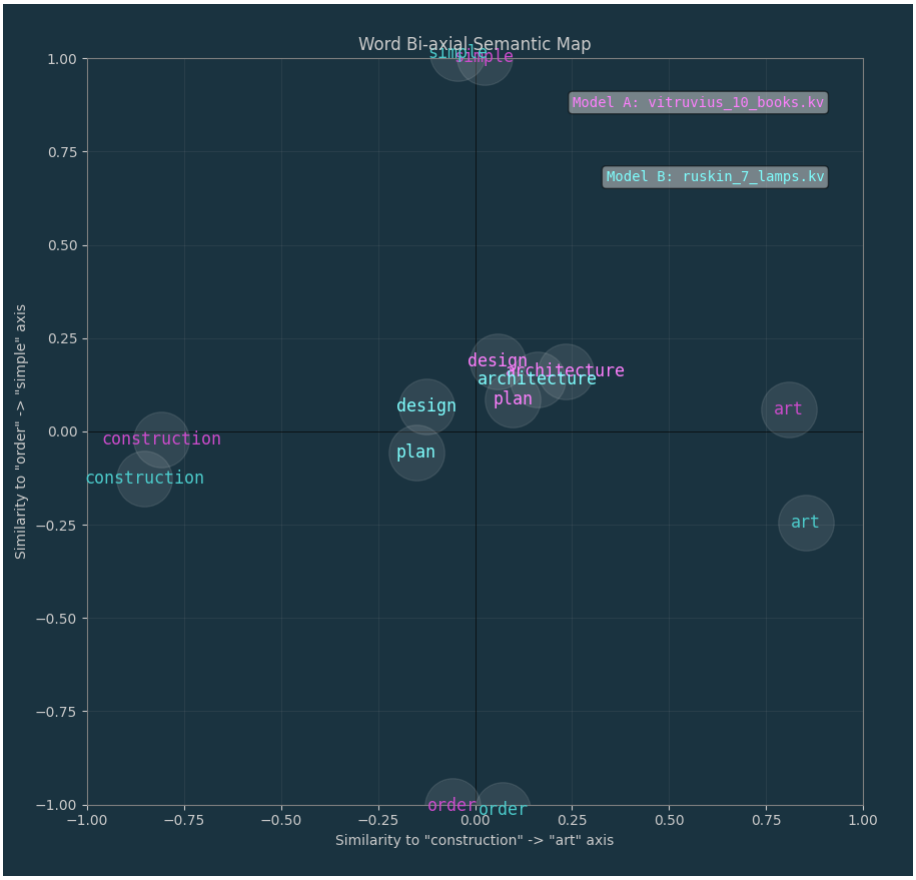
Add a screenshot of the visualization to your assignment submission.

For the next step we need to modify the script so that its input words are selected from the words close to a central term for each model.

So select a word (e.g. "architecture"). We will call it the central_word

Now the list of words_to_plot should consist of the central_word along with its 5 most similar words from models a and b (10 words in total)

In my case the diagram looks like this:



One thing to pay attention to is that when we vectorize and plot each word from the words to plot in the loop:

```
for word in words_to_plot:
    #get the normalized vector for the word
    v_a = model_a.get_vector(word, norm=True)
    v_b = model_b.get_vector(word, norm=True)
```

the get_vector operation can fail because a word that exists in one text may not exist in another. To make this work we need to use a conditional statement (“if”), so a word is plotted for a model only if it exists for that model. We can use the special test keyword “in” which test whether a “key” exists in a container.

```
    if word in model_a:
        #do something here
        v_a = model_a.get_vector(word, norm=True)
```