NLP: Embeddings

Welcome to the embedding portion of this session!

Part 1: word2vec word embeddings

We will start off by using word2vec to take words and embed them into a vector space.

In [53]:
```
import spacy

# Load the spaCy model
word_embedder = spacy.load("en_core_web_lg")

# Create a test sentence
sentence = "hello world word embedding"

# This is how spaCy wants to embed words
embedded_sentence = word_embedder(sentence)

# Here we can see the embedding for this first word
print(embedded_sentence[0].vector)
```

```
[ 0.25233    0.10176   -0.67485    0.21117    0.43492    0.16542
  0.48261   -0.81222    0.041321   0.78502   -0.077857  -0.66324
  0.1464    -0.29289   -0.25488    0.019293  -0.20265    0.98232
  0.028312  -0.081276  -0.1214     0.13126   -0.17648    0.13556
 -0.16361   -0.22574    0.055006  -0.20308    0.20718    0.095785
  0.22481    0.21537   -0.32982   -0.12241   -0.40031   -0.079381
 -0.19958   -0.015083  -0.079139  -0.18132    0.20681   -0.36196
 -0.30744   -0.24422   -0.23113    0.09798    0.1463    -0.062738
  0.42934   -0.078038  -0.19627    0.65093   -0.22807   -0.30308
 -0.12483   -0.17568   -0.14651    0.15361   -0.29518    0.15099
 -0.51726   -0.033564  -0.23109   -0.7833     0.018029  -0.15719
  0.02293    0.49639    0.029225   0.05669    0.14616   -0.19195
  0.16244    0.23898    0.36431    0.45263    0.2456     0.23803
  0.31399    0.3487    -0.035791   0.56108   -0.25345    0.051964
 -0.10618   -0.30962    1.0585    -0.42025    0.18216   -0.11256
  0.40576    0.11784   -0.19705   -0.075292   0.080723  -0.02782
 -0.15617   -0.44681   -0.15165    0.1692     0.098255  -0.031894
  0.087143   0.26082    0.002706   0.1319     0.34439   -0.37894
 -0.4114     0.081571  -0.11674   -0.43711    0.011144   0.099353
  0.26612    0.40025    0.18895   -0.18438   -0.30355   -0.2725
  0.22468   -0.40614    0.15618   -0.16043    0.47147    0.0080203
  0.56858    0.21934   -0.11181    0.79925    0.10714   -0.50146
  0.063593   0.069465   0.15292   -0.2747    -0.20989    0.20737
 -0.10681    0.40651   -2.6438    -0.31139   -0.32157   -0.26458
 -0.35625    0.070013  -0.18838    0.48773   -0.26167   -0.020805
  0.17819    0.15758   -0.13752    0.056464   0.30766   -0.066136
  0.4748    -0.27335    0.09732   -0.20832    0.0039332  0.346
 -0.08702   -0.54924   -0.18759   -0.17174    0.060324  -0.13521
  0.10419    0.30165    0.05798    0.21872   -0.073594  -0.20423
 -0.25279   -0.10471   -0.32163    0.12525   -0.31281    0.0097207
 -0.26777   -0.61121   -0.11089   -0.13652    0.035135  -0.4939
  0.084857  -0.15494   -0.063509  -0.23935    0.28272    0.10849
 -0.3365    -0.60764    0.38576   -0.0095438  0.17499   -0.52723
  0.62211    0.19544   -0.48977    0.036582  -0.128     -0.016827
  0.25647   -0.31698    0.48257   -0.14184    0.11046   -0.3098
 -0.63141   -0.37268    0.23183   -0.14268   -0.02341    0.022255
 -0.044662  -0.16404   -0.25848    0.1629     0.024751   0.23348
  0.27933    0.38998   -0.058968   0.11355    0.15673    0.18583
 -0.19814   -0.48123   -0.035084   0.078458  -0.49833    0.10855
 -0.20133    0.05292   -0.11583   -0.16009    0.16768    0.42362
 -0.23106    0.082465   0.24296   -0.16786    0.0080409  0.085947
  0.38033    0.072981   0.1633     0.24704   -0.11094    0.15115
```

```
-0.22068   -0.061944  -0.037091  -0.087923  -0.23181    0.15035
-0.19093   -0.19113   -0.11894    0.094908  -0.0043347  0.15362
-0.41201   -0.3073     0.18375    0.40206   -0.0034793 -0.10917
-0.69522    0.10161   -0.079256   0.40329    0.22285   -0.19374
-0.13315    0.073231   0.099832   0.11685   -0.21643   -0.1108
 0.10341    0.097286   0.11196   -0.3894    -0.0089363  0.28809
-0.10792    0.028811   0.32545    0.26052   -0.038941   0.075204
 0.46031   -0.06293    0.21661    0.17869   -0.51917    0.33591   ]
```

Word Embeddings, how can we use them?

Here we will use the distance in the embeddings space to find the most similar word in a list

Feel free to play around with the word list and the example word to see how these interact with each other

In [ ]:
```
import numpy as np

# Generate a list of 100 random words
random_words = "acorn breeze candle drift ember falcon glint harbor ink jumble kernel latch mirth nudge orbit prism c

# Embed the random words
random_word_embeddings = word_embedder(random_words)

# Define a function to find the most similar word
def find_most_similar_word(word, words_to_match, word_embedding_tool):
    # Embed the input word
    word_embedding = word_embedding_tool(word)

    # Embeddings from word_list
    random_word_embeddings = word_embedding_tool(random_words)

    # Compute cosine similarity
    similarities = []
    for i in range(len(random_word_embeddings)):
        similarity = word_embedding.similarity(random_word_embeddings[i])
        similarities.append(similarity)

    # This is our most similar word
    most_similar_index = np.argmax(similarities)

    # Return the most similar word by splitting to to match string
```

```
    match_list = words_to_match.split(' ')

    return match_list[most_similar_index]

# Example usage
input_word = "pen"
most_similar_word = find_most_similar_word(input_word, random_words, word_embedder)
print(f"The most similar word to '{input_word}' is '{most_similar_word}'.")
```

```
The most similar word to 'pen' is 'ink'.
```

```
/var/folders/lz/dxxzxhj966zbjy685vyxqcl00000gp/T/ipykernel_77158/1267569091.py:20: UserWarning: [W008] Evaluating Do
c.similarity based on empty vectors.
  similarity = word_embedding.similarity(random_word_embeddings[i])
```

A cool property about word2vec embeddings

Here we are going to see how the distances in these embeddings themselves have meaning. The implication here is that there is some dimenson-like element that encodes certain properties

In [76]:

```
def compute_distance(word_1, word_2, word_embedding_tool):
    # Embed the words
    word_1_embedding = word_embedding_tool(word_1)
    word_2_embedding = word_embedding_tool(word_2)

    # Compute the distance
    distance = word_1_embedding.vector - word_2_embedding.vector

    return distance

# Embed the royal words
royal_difference = compute_distance("king", "queen", word_embedder)

# Embed kid words
kid_difference = compute_distance("boy", "girl", word_embedder)

# Embed some random words
random_difference = compute_distance("spaceship", "dog", word_embedder)

# Create difference of differences
def cosine_similarity(a, b):
```

```
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))

# Compute the cosine similarity between the two differences
royal_kid_similarity = cosine_similarity(royal_difference, kid_difference)
print(f"The cosine similarity between the kid and royal differences is: {royal_kid_similarity}")

# Compute the cosine similarity between the two differences
royal_random_similarity = cosine_similarity(royal_difference, random_difference)
print(f"The cosine similarity between the random and royal differences is: {royal_random_similarity}")

# Finally, show the cosine similarity between the two royal words
royal_similarity = cosine_similarity(word_embedder("king")[0].vector, word_embedder("queen")[0].vector)
print(f"The cosine similarity between the royal words is: {royal_similarity}")
```

```
The cosine similarity between the kid and royal differences is: 0.475158035755l5747
The cosine similarity between the random and royal differences is: -0.0818493589758873
The cosine similarity between the royal words is: 0.7252610921859741
```

Part 2: Sentence Embeddings

Now we will use small transformer models (what state of the art models like ChatGPT use!) to play with sentence embeddings

In [ ]:
```
from sentence_transformers import SentenceTransformer, util
import torch

# Load a pre-trained sentence transformer model
sentence_model = SentenceTransformer('all-MiniLM-L6-v2')

# List of sentences to embed
sentences = [
    "The quick brown fox jumps over the lazy dog.",
    "A journey of a thousand miles begins with a single step.",
    "To be or not to be, that is the question.",
    "All that glitters is not gold.",
    "The pen is mightier than the sword."
]

# Embed the sentences
sentence_embeddings = sentence_model.encode(sentences, convert_to_tensor=True)

# New example sentence
```

```python
example_sentence = "A long journey starts with one step."

# Embed the example sentence
example_embedding = sentence_model.encode(example_sentence, convert_to_tensor=True)

# Compute cosine similarities
cosine_similarities = util.cos_sim(example_embedding, sentence_embeddings)

# Find the most similar sentence
most_similar_idx = torch.argmax(cosine_similarities).item()
most_similar_sentence = sentences[most_similar_idx]

print(f"The most similar sentence to '{example_sentence}' is '{most_similar_sentence}'.")
```

```
The most similar sentence to 'A long journey starts with one step.' is 'A journey of a thousand miles begins with a s
ingle step.'.
```

In [92]:

```python
from datasets import load_dataset

# Now load a much larger dataset of sentences
ds = load_dataset("agentlans/high-quality-english-sentences")

# Load a pre-trained sentence transformer model
sentence_model = SentenceTransformer('all-MiniLM-L6-v2')

# Embed the first 10,000 sentences (this is a subset to run faster)
sentence_embeddings = sentence_model.encode(ds['train'][:10000]['text'], convert_to_tensor=True)
```

We split out the comparison cell because the embedding of 10k sentences can take a little time

In [93]:

```python
# New example sentence
example_sentence = "A long journey starts with one step."
# Embed the example sentence
example_embedding = sentence_model.encode(example_sentence, convert_to_tensor=True)
# Compute cosine similarities
cosine_similarities = util.cos_sim(example_embedding, sentence_embeddings)
# Find the most similar sentence
most_similar_idx = torch.argmax(cosine_similarities).item()
most_similar_sentence = ds['train'][most_similar_idx]['text']
print(f"The most similar sentence to '{example_sentence}' is '{most_similar_sentence}'.")
```

```
The most similar sentence to 'A long journey starts with one step.' is 'Journeys are a very important part of our fai
th tradition, too.'.
```