Word Embeddings:

Word2Vec transforms words into dense vectors, where semantically similar words are represented by similar vectors. For example, the words "king" and "queen" will have vectors that are close together in the embedding space. Training Methods

Continuous Bag fo Words (CBOW):

Predicts a target word from its surrounding context words. Given a set of context words, CBOW tries to find the most probable word that fits that context.

- Skip-gram: Predicts the context words given a target word. It focuses on the relationships between words by attempting to maximize the probability of context words given a specific word.

High-Dimensional Space:

Word vectors are typically represented in a high-dimensional space (e.g., 100 to 300 dimensions). The higher the dimensions, the more information the vectors can potentially capture, but it also increases computational complexity.

Similarity and Analogy:

Word2Vec allows for measuring the similarity between words using cosine similarity. It also enables performing analogies, such as "king - man + woman = queen."

Advantages of Word2Vec

Efficiency: It can efficiently process large datasets and learn high-quality word embeddings. Capturing Semantic Relationships: Word2Vec captures semantic relationships between words better than traditional one-hot encoding.

Transfer Learning:

Pre-trained Word2Vec embeddings can be fine-tuned for specific tasks, saving time and computational resources.

Limitations of Word2Vec

- Static Representations: Word2Vec generates static embeddings, meaning that each word has a single vector representation regardless of context. This can lead to issues with polysemy (words with multiple meanings).
- Out-of-Vocabulary (OOV) Words: Words that were not in the training dataset will not have embeddings, limiting the model's ability to generalize.

In [1]:
```
import gensim
from gensim.models import Word2Vec

# Sample sentences for training
sentences = [
    ["I", "love", "natural", "language", "processing"],
    ["Word2Vec", "is", "a", "great", "tool"],
    ["Machine",  "learning", "is", "fun"],
]

# Train the Word2Vec model
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, sg=1)

# Get the vector for a word
vector = model.wv['language']
print("Vector for 'Machine':", vector)

# Find similiar words
similar_words = model.wv.most_similar('language', topn=1)
print("Words similar to 'Machine':", similar_words)
```

```
Vector for 'Machine': [-0.00515624 -0.00666834 -0.00777684  0.00831073 -0.00198234 -0.00685496
 -0.00415439  0.00514413 -0.00286914 -0.00374966  0.00162143 -0.00277629
 -0.00158436  0.00107449 -0.00297794  0.00851928  0.00391094 -0.00995886
  0.0062596  -0.00675425  0.00076943  0.00440423 -0.00510337 -0.00211067
  0.00809548 -0.00424379 -0.00763626  0.00925791 -0.0021555  -0.00471943
  0.0085708   0.00428334  0.00432484  0.00928451 -0.00845308  0.00525532
  0.00203935  0.00418828  0.0016979   0.00446413  0.00448629  0.00610452
 -0.0032021  -0.00457573 -0.00042652  0.00253373 -0.00326317  0.00605772
  0.00415413  0.00776459  0.00256927  0.00811668 -0.00138721  0.00807793
  0.00371702 -0.00804732 -0.00393361 -0.00247188  0.00489304 -0.00087216
 -0.00283091  0.00783371  0.0093229  -0.00161493 -0.00515925 -0.00470176
 -0.00484605 -0.00960283  0.00137202 -0.00422492  0.00252671  0.00561448
 -0.00406591 -0.00959658  0.0015467  -0.00670012  0.00249517 -0.00378063
  0.00707842  0.00064022  0.00356094 -0.00273913 -0.00171055  0.00765279
  0.00140768 -0.00585045 -0.0078345   0.00123269  0.00645463  0.00555635
 -0.00897705  0.00859216  0.00404698  0.00746961  0.00974633 -0.00728958
 -0.00903996  0.005836    0.00939121  0.00350693]
Words similar to 'Machine': [('is', 0.21617141366004944)]
```

In [2]:
```
pip install gensim
```

```
Requirement already satisfied: gensim in c:\users\samua\anaconda3\envs\tensorflow_env\lib\site-packages (4.3.3)
Requirement already satisfied: numpy<2.0,>=1.18.5 in c:\users\samua\anaconda3\envs\tensorflow_env\lib\site-packages (from gensi
m) (1.26.4)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in c:\users\samua\anaconda3\envs\tensorflow_env\lib\site-packages (from gen
sim) (1.13.1)
Requirement already satisfied: smart-open>=1.8.1 in c:\users\samua\anaconda3\envs\tensorflow_env\lib\site-packages (from gensi
m) (7.3.0.post1)
Requirement already satisfied: wrapt in c:\users\samua\anaconda3\envs\tensorflow_env\lib\site-packages (from smart-open>=1.8.1-
>gensim) (1.17.0)
Note: you may need to restart the kernel to use updated packages.
```

Sumnmary

Word2Vec is a foundational technique in NLP that facilitates the understanding of word semantics through vector representations. Despite its limitations, it has paved the way for more advanced embedding techniques, such a GloVe and contextual embeddings like BERT and ELMo.

CBOW & SKIPGRAM

CBOW (Continuous Bag of words) and Skip-gram are two architectures used in Word2Vec for generating word embeddings. Both approaches aim to learn vector representations of words based on their context, but they do so in different waves.

In [ ]:

In [ ]: