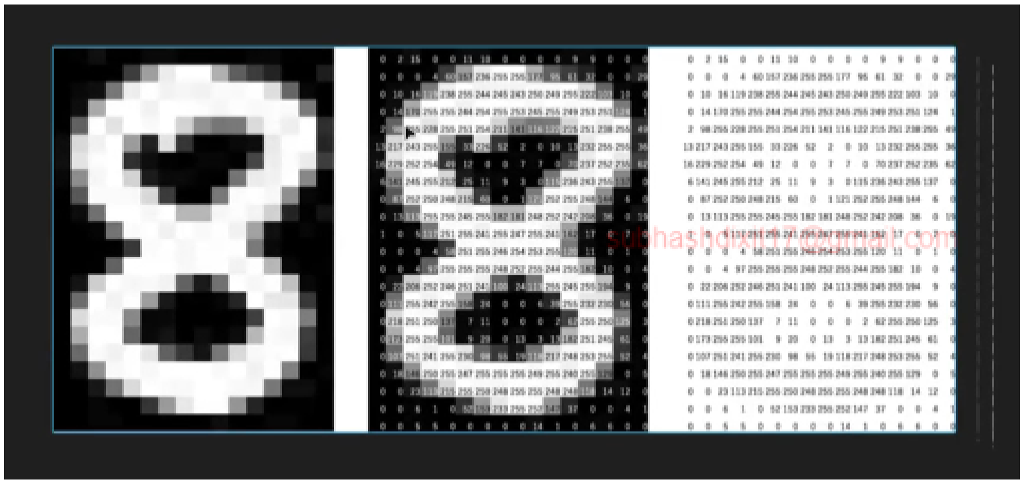
NLP: Day 5

Word Embedding:

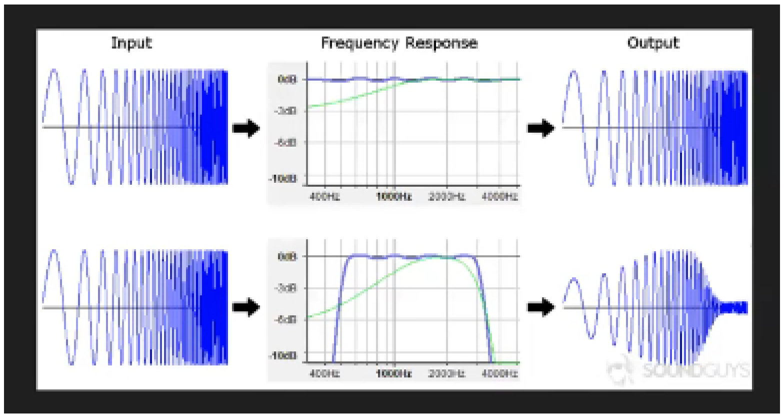
- Word embedding represents words as numerical vectors in a dense vector space.
- It captures the meaning and relationships between words.
- Examples include representing words like "cat" and "dog" as vectors.
- Word embeddings are used in NLP tasks like text classification and machine translation.
- They help understand and process textual data more effectively.

Text Representation:

- Text-to numbers/vectors conversion is essential in machine learning to represent text data in a format that can be processed by algorithms.
- Deep learning models like transformers and BERT don't require explicit text to numbers conversion as they have built-in embedding layers.
- Word embedding techniques are used to convert words into numerical representations:
- One-hot encoding assigns a binary vector to each word based on its position in the vocabulary.
- Bag of words represents text as a collection of word occurrences, disregarding word order.
- N-grams capture sequences of N words in the text.
- TF-IDF (Term Frequency-Inverse Document Frequency) calculates the importance of a word in a document relative to a corpus.
- Keras Embedding layer learns word embeddings during model training.
- Word2Vec is a popular deep-learning technique for learning word embeddings.
- CBOW (Continuous Bag of Words) predicts a target word given the surrounding context words.
- Skip-grams predict context words based on a target word.
- **Image data** can be represented as pixel values, where each pixel has a specific intensity or colour. 0 px – Represent black, 155 px –Represent White



- **Audio data** is represented by frequencies, and the amplitude of frequencies at different time intervals can be stored.



- Feature extraction from text involves transforming raw text into meaningful numerical features.
- **Example:** "My name is Subhash" - Raw text
- **Key points**:
    - Feature extraction is necessary to represent text data in a format suitable for analysis and modelling.
    - Various techniques like one-hot encoding, bag of words, N-grams, TF-IDF, word embeddings, and others are used.
    - Image data is represented by pixels, audio data by frequencies, and text data requires feature extraction for numerical representation.

Corpus:

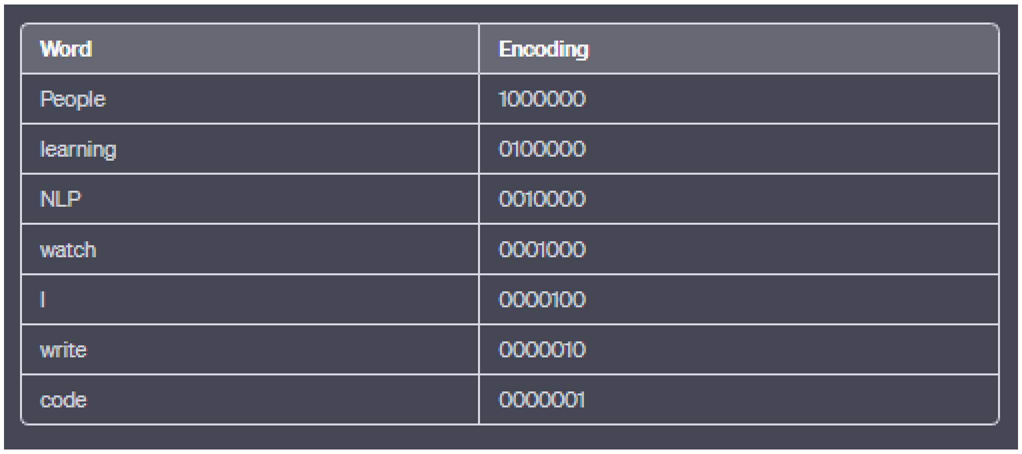
- A corpus is a collection of text documents used for analysis or modelling.
- The given corpus consists of four sentences: "People learning NLP", "NLP watch NLP", "I write code", and "People write code".
- The vocabulary is the set of unique words present in the corpus.
- In this case, the vocabulary contains the following words: "People", "Learning", "NLP", "watch", "I", "write", and "code".
- The size of the vocabulary is 7, indicating that there are 7 unique words in the corpus.
- **Key points:**
    - A corpus is a collection of text documents.
    - The **vocabulary** represents the unique words in the corpus.
    - The size of the vocabulary indicates the number of unique words in the corpus.

Word Embedding Types:

1. **One-hot encoding:**
    - One-Hot Encoding is a technique used to represent categorical data as binary vectors.
    - Each word in the vocabulary is assigned a unique binary vector where only one element is 1 (hot) and the rest are 0 (not hot).
    - The length of the binary vectors is equal to the vocabulary size.

- One-Hot Encoding is commonly used in Natural Language Processing (NLP) tasks for representing text data.
- It treats each word independently and does not capture the semantic relationships between words.
- One-Hot Encoding is a simple and straightforward method but can lead to high-dimensional and sparse representations, especially with large vocabularies.
- **Advantages of One-Hot Encoding:**
    - **Easy Implementation:** One-Hot Encoding is simple to implement and understand.
    - **Preserves Categorical Information:** It represents each category as a separate binary vector, preserving the categorical information in the data.
    - **Compatibility with Machine Learning Algorithms:** One-Hot Encoding is compatible with various machine learning algorithms that expect numerical inputs.
- **Disadvantages of One-Hot Encoding:**
    - **High Dimensionality:** One-Hot Encoding can result in high-dimensional feature spaces, especially with large vocabularies, which may lead to memory and computational challenges.
    - **Sparsity:** One-Hot Encoding produces sparse representations, as most words are encoded as 0s in the binary vectors, which can increase storage requirements and impact efficiency.
    - **Ignores Semantic Relationships:** One-Hot Encoding treats each word independently and does not capture semantic relationships between words, leading to loss of contextual information.
- **Note:** One-Hot Encoding is suitable for certain tasks and datasets, but it may not be the most efficient or effective encoding method in all cases. Other techniques such as word embeddings or more advanced encoding schemes can be used to address the limitations of One-Hot Encoding.
- **Steps to Calculate One-Hot Encoded Vectors:**
    - **Example - Corpus:**
        - Document 1: People learning NLP
        - Document 2: NLP watch NLP
        - Document 3: I write code
        - Document 4: People write code
    - **Create a vocabulary:** Identify all unique words in the sentences.
    - **Vocabulary:** People, learning, NLP, watch, I, write, code
    - Assign an index to each word in the vocabulary.
        - People: 1
        - learning: 2
        - NLP: 3
        - watch: 4
        - I: 5
        - write: 6
        - code: 7

- Create a binary vector for each word using one-hot encoding.
  - People: 1000000
  - learning: 0100000
  - NLP: 0010000
  - watch: 0001000
  - I: 0000100
  - write: 0000010
  - code: 0000001



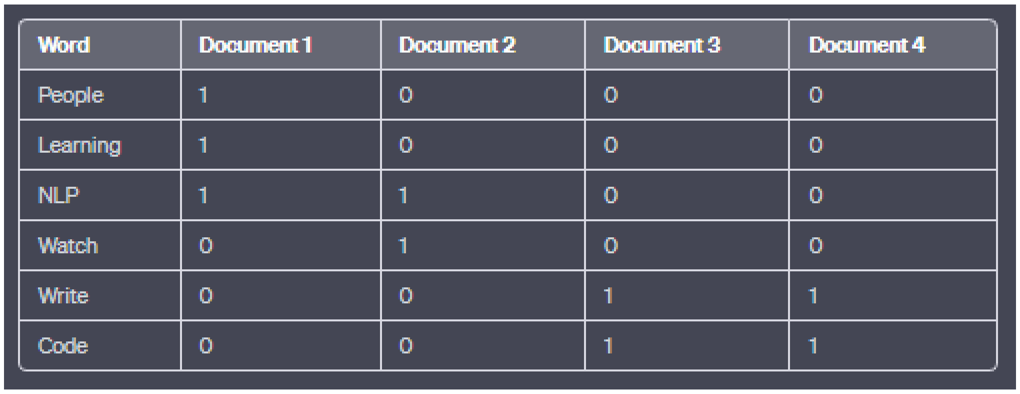
Word	Encoding
People	1000000
learning	0100000
NLP	0010000
watch	0001000
I	0000100
write	0000010
code	0000001

- **Note**: Each row in the table represents a word from the sentences, and the encoding column shows the corresponding one-hot encoded vector. The encoding vectors have a length equal to the size of the vocabulary, and each vector contains a single '1' at the index corresponding to the word's position in the vocabulary.

2. **Bag of Words:**
   - The Bag of Words (BoW) is a common technique used in Natural Language Processing (NLP) to represent text data as a collection of words and their frequencies. It treats each document as an unordered collection of words, disregarding grammar and word order.
   - **Advantages:**
     - **Simple and easy to implement:** The Bag of Words model is straightforward to implement and understand. It does not require complex algorithms or extensive preprocessing steps.
     - **Versatile representation:** BoW representation can be used for various NLP tasks such as text classification, sentiment analysis, and information retrieval.
     - **Preserves semantic meaning:** Despite disregarding word order, the frequency of words still provides some information about their importance and relevance in a document.

    - **Language independence:** BoW representation can be applied to texts in any language without requiring language-specific preprocessing.
- **Disadvantages:**
    - **Ignores word relationships:** BoW representation does not consider the context or relationships between words in a document, resulting in a loss of sequential and semantic information.
    - **Large feature space:** As the vocabulary size increases, the dimensionality of the feature space grows, which can lead to the "curse of dimensionality" and increased computational complexity.
    - **No consideration of word importance:** BoW treats all words equally, regardless of their significance or meaning. This can result in important words being overshadowed by frequent but less meaningful ones.
    - **Limited understanding of semantics:** BoW representation cannot capture the nuances of language, such as sarcasm, ambiguity, or wordplay, as it only focuses on word frequencies.
- **Steps to Calculate Bag of Words:**
    - **Tokenization:** Split the text into individual words or tokens. This can be done using whitespace, punctuation, or more advanced tokenization techniques.
    - **Lowercasing:** Convert all words to lowercase to ensure consistent representation and avoid treating the same word as different due to case differences.
    - **Removing Stop Words:** Remove common words that do not contribute much to the overall meaning of the text, such as articles, prepositions, and conjunctions.
    - **Building Vocabulary:** Create a unique set of words from the remaining tokens. This set will serve as the vocabulary or feature space for the Bag of Words representation.
    - **Word Frequency Count:** Count the frequency of each word in the text. This can be done by iterating through the tokens and updating the count for each word in the vocabulary.
    - **Vectorization:** Represent each document as a vector, where each element corresponds to the frequency of a word in the vocabulary. The vector will have the same length as the vocabulary size.
    - **Normalization:** Optionally, normalize the vector representation to account for document length or to apply weighting schemes such as TF-IDF (Term Frequency-Inverse Document Frequency).
    - **Note:** These steps provide a basic overview of the Bag of Words calculation process. Additional preprocessing and techniques can be applied depending on the specific requirements of the task or application.
    - **Example:**
        - **Corpus:**
            - Document 1: People learning NLP
            - Document 2: NLP watch NLP

- Document 3: I write code
- Document 4: People write code



Word	Document 1	Document 2	Document 3	Document 4
People	1	0	0	0
Learning	1	0	0	0
NLP	1	1	0	0
Watch	0	1	0	0
Write	0	0	1	1
Code	0	0	1	1

Advantages:

- By word count, we can emphasize more on the word. If the count of good words is more then we can say positive review. It is good for sentiment analysis problems.

Disadvantages:

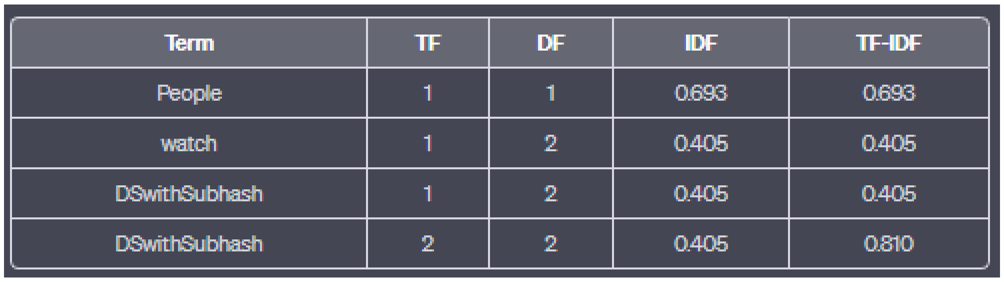
- Sparse matrix problems still exist

3. **Tf-IDF:**
    - Term Frequency (TF) measures the frequency of a term within a document.
    - Document Frequency (DF) measures the number of documents that contain a particular term.
    - Inverse Document Frequency (IDF) measures the informativeness of a term across multiple documents.
    - TF-IDF combines the TF and IDF to determine the importance of a term within a specific document.
    - The TF-IDF word embedding table shows the TF, DF, IDF, and TF-IDF values for each term in the given sentences.
    - The higher the TF-IDF value, the more significant or unique the term is within the document.
    - **Advantages of TF-IDF:**
        - Term Importance: Identifies key informative terms within a document.
        - Handles Stop Words: Naturally handles common stop words.
        - Domain Specificity: Incorporates domain knowledge by assigning unique scores.
        - Flexibility: Can be customized and adjusted for specific requirements.
        - Widely Used: Established and widely adopted in various applications.
    - **Disadvantages of TF-IDF:**
        - Lack of Semantic Understanding: Does not capture semantic relationships between words.

- Vocabulary Size: Requires maintaining a large vocabulary for complex datasets.
    - Document Length Bias: Longer documents can bias term frequency.
    - Sensitivity to Term Variations: Treats similar terms as separate entities.
    - Lack of Document Coherence: Ignores document structure and context.
- **Note:** TF-IDF remains valuable but can be enhanced when used with other techniques or models.

- ***To create a TF-IDF (Term Frequency-Inverse Document Frequency) word embedding table for the given sentences, we'll follow these steps:***
- ***R1*** *→ People watch DSwithSubhash*
- ***R2*** *→ DSwithSubhash watch DSwithSubhash*
- **Step 1: Calculate Term Frequency (TF):**
    - Term Frequency (TF) measures the frequency of a term within a document.
    - For Sentence R1:
        - TF("People") = 1
        - TF("watch") = 1
        - TF("DSwithSubhash") = 1
    - For Sentence R2:
        - TF("DSwithSubhash") = 2
        - TF("watch") = 1
- **Step 2: Calculate Document Frequency (DF):**
    - Document Frequency (DF) measures the number of documents that contain a particular term.
    - For the given sentences:
        - DF("People") = 1
        - DF("watch") = 2
        - DF("DSwithSubhash") = 2
- **Step 3: Calculate Inverse Document Frequency (IDF):**
    - Inverse Document Frequency (IDF) measures the informativeness of a term across multiple documents.
        - $IDF\, formula: IDF(t) = log(N / (1 + DF(t)))$
        - N represents the total number of documents in the corpus or dataset
    - For the given sentences:
        - IDF("People") = log(2 / (1 + 1)) = log(2) ≈ 0.693
        - IDF("watch") = log(2 / (1 + 2)) = log(0.666) ≈ 0.405
        - IDF("DSwithSubhash") = log(2 / (1 + 2)) = log(0.666) ≈ 0.405
- **Step 4: Calculate TF-IDF**:
    - TF-IDF combines the TF and IDF to determine the importance of a term within a specific document.
        - $TF - IDF\, formula: TF - IDF(t,\, d) = TF(t,\, d) * IDF(t)$

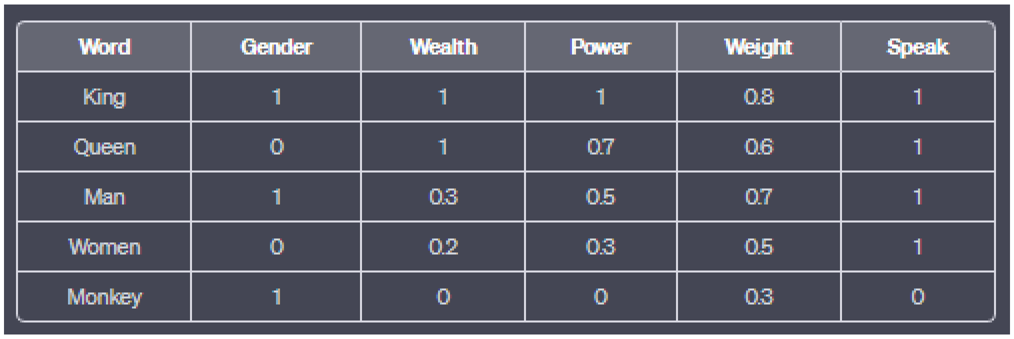
- TF-IDF Word Embedding Table:



Term	TF	DF	IDF	TF-IDF
People	1	1	0.693	0.693
watch	1	2	0.405	0.405
DSwithSubhash	1	2	0.405	0.405
DSwithSubhash	2	2	0.405	0.810

4. **NGrams:**
   - N-grams are contiguous sequences of n items, typically used in the context of natural language processing. They help analyze the structure and relationships between words in a sentence or a corpus of text. The types of N-grams commonly used are unigrams (n=1), bigrams (n=2), trigrams (n=3), and so on.
   - **Example:** Sentence 1: "Alex is a bday boy"
     - Unigrams: ['Alex', 'is', 'a', 'bday', 'boy']
     - Bigrams: ['Alex is', 'is a', 'a bday', 'bday boy']
     - Trigrams: ['Alex is a', 'is a bday', 'a bday boy']
   - **Example:** Sentence 2: "Alex is not a bad boy"
     - Unigrams: ['Alex', 'is', 'not', 'a', 'bad', 'boy']
     - Bigrams: ['Alex is', 'is not', 'not a', 'a bad', 'bad boy']
     - Trigrams: ['Alex is not', 'is not a', 'not a bad', 'a bad boy']
   - In BOW, word count is focused more so “not” will not have more importance
   - In NGram, “not” will not be missed out

5. **Word2Vec:**
   - Word2Vec is a popular word embedding technique used in NLP.
   - It represents words as vectors, capturing semantic relationships.
   - **Advantages:**
     - Captures semantic similarities.
     - Dimensionality reduction.
     - Generalization of word meanings.
   - **Disadvantages:**
     - Lack of interpretability.
     - Contextual limitations.
     - Difficulty with out-of-vocabulary words.
   - Widely used in NLP tasks like sentiment analysis and document classification.
   - Pre-trained models and domain-specific training can enhance performance.
   - **Note:** Word2Vec is a powerful word embedding technique with advantages in capturing semantic relationships, but it has limitations in interpretability and context. Other techniques like GloVe and fastText should be considered based on the specific task and data characteristics.
   - **Example:**

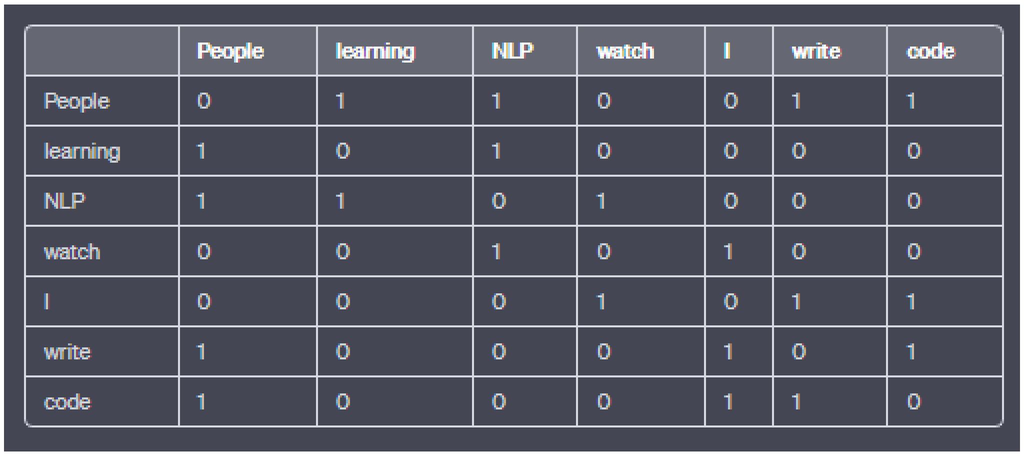


Word	Gender	Wealth	Power	Weight	Speak
King	1	1	1	0.8	1
Queen	0	1	0.7	0.6	1
Man	1	0.3	0.5	0.7	1
Women	0	0.2	0.3	0.5	1
Monkey	1	0	0	0.3	0

- The weight values in the table indicate the degree of association between each word and the concept of weight.
- Words with higher weight values, such as "King," "Queen," and "Man," suggest a stronger association or relevance to weight.
- On the other hand, words with lower weight values, like "Women" and "Monkey," indicate a relatively weaker association with weight.
- The weight values in the table provide insights into how weight is semantically linked to different words in the given context.
- **Note:** The weight values reflect the relative importance or relevance of weight as a feature and should not be interpreted as actual physical weights. They are derived from the word embeddings generated by the Word2Vec model, capturing the semantic relationships between words.

6. **GloVe Word:**
    - GloVe (Global Vectors for Word Representation) is a word embedding technique developed by Stanford researchers.
    - It combines global matrix factorization and local context window methods to generate word vectors.
    - **Advantages:**
        - Captures semantic relationships between words.
        - Handles rare and frequent words well.
        - Can be pre-trained on large corpora and used for transfer learning.
    - **Disadvantages:**
        - Requires a large amount of training data to generate accurate embeddings.
        - May struggle with capturing complex semantic nuances.
        - Limited to representing individual words and not phrases or sentences.
    - **Steps to calculate Glove Word embedding:**
        - **Corpus:**
            - Document 1: People learning NLP
            - Document 2: NLP watch NLP
            - Document 3: I write code
            - Document 4: People write code

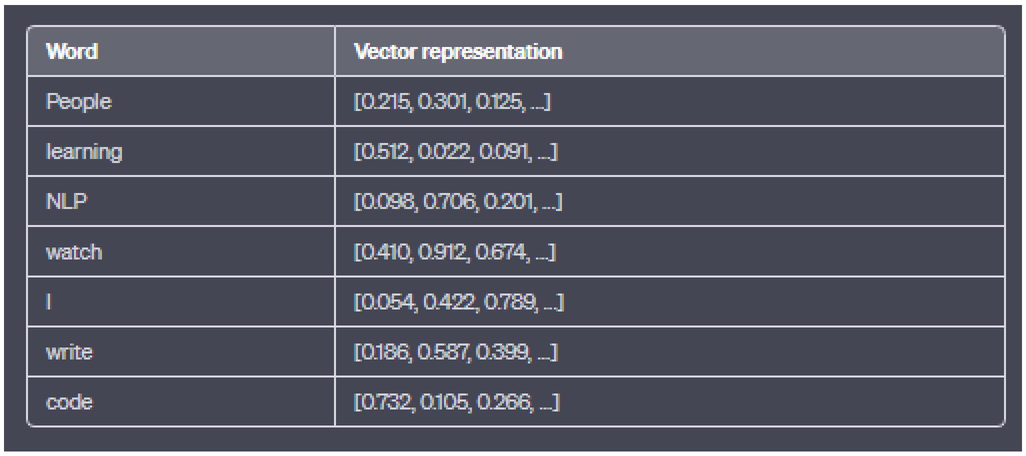
- **Build a corpus:** Combine all the documents to create a corpus of text.
- **Tokenization:** Split the text into individual words or tokens.
- **Create a co-occurrence matrix:** Count the number of times each word co-occurs with other words within a specified window size.
  - Build a corpus: Combine all the documents to create a corpus of text.
  - Tokenization: Split the text into individual words or tokens.
  - Define a context window: Determine the size of the context window, which specifies the number of words to consider around each target word.
  - Initialize the co-occurrence matrix: Create an empty matrix with rows and columns representing the unique words in the vocabulary.
  - Iterate through the documents: For each document, iterate through the words.
  - Determine the context words: For each target word, identify the context words within the specified window size.
  - Update the co-occurrence matrix: Increment the count of co-occurrences between the target word and each context word in the matrix.
  - Repeat for all documents: Repeat steps 5-7 for all documents in the corpus.
  - Finalize the co-occurrence matrix: The resulting matrix represents the co-occurrence counts between words.



	People	learning	NLP	watch	I	write	code
People	0	1	1	0	0	1	1
learning	1	0	1	0	0	0	0
NLP	1	1	0	1	0	0	0
watch	0	0	1	0	1	0	0
I	0	0	0	1	0	1	1
write	1	0	0	0	1	0	1
code	1	0	0	0	1	1	0

- **Calculate word probabilities:** Convert the co-occurrence matrix into probabilities by dividing each count by the total count of co-occurrences for that word.
- **Initialize word vectors:** Initialize word vectors for each word in the vocabulary.
- **Define the loss function:** Define a loss function that measures the difference between the predicted word vector and the target word vector.

- **Train the word vectors:** Use an optimization algorithm, such as gradient descent, to update the word vectors based on the loss function.
- **Extract the learned word vectors:** After training, extract the learned word vectors for each word in the vocabulary.



Word	Vector representation
People	[0.215, 0.301, 0.125, ...]
learning	[0.512, 0.022, 0.091, ...]
NLP	[0.098, 0.706, 0.201, ...]
watch	[0.410, 0.912, 0.674, ...]
I	[0.054, 0.422, 0.789, ...]
write	[0.186, 0.587, 0.399, ...]
code	[0.732, 0.105, 0.266, ...]

- **Note:** The vector representations in the table are example values and may not reflect the actual GloVe word vectors for the given documents