Text Encoding and Embedding:

Why Encoding & Embedding?

This is the stage where raw text (now preprocessed) is converted into numerical form so machine learning or deep learning models can process it. It's critical for both classical and Generative AI (LLM) models.

Machines don’t understand text, only numbers. So we must convert:

"I love NLP" → [vectors or matrices]

The goal is to represent the meaning of text while preserving semantic similarity and syntactic structure.

1. **Text Encoding Techniques (Classical)**

1.1 Bag of Words (BoW)

- Treats text as a "bag" of words, ignoring order and grammar.
- Each document is represented as a **vector of word counts**.
- Works well for tasks like spam detection or topic classification.

Example:

Let’s say we have 2 documents:

Doc1: "I love NLP"

Doc2: "NLP is fun"

Vocabulary = [I, love, NLP, is, fun]

Term	Doc1	Doc2
I	1	0
love	1	0
NLP	1	1
is	0	1
fun	0	1

Code:

```
from sklearn.feature_extraction.text import CountVectorizer


docs = ["I love NLP", "NLP is fun"]
vectorizer = CountVectorizer()


bow = vectorizer.fit_transform(docs)
print("Vocabulary:", vectorizer.get_feature_names_out())
print("BoW Matrix:\n", bow.toarray())
```

1.2 TF-IDF (Term Frequency–Inverse Document Frequency)

- Improves upon BoW by considering **importance** of a word in context of a corpus.
- Formula:

TF = (term count in doc) / (total terms in doc)

IDF = log(N / df), where df = doc frequency

TF-IDF = TF × IDF

Example:

- Words like “the” appear often and aren’t informative → they get low weight.
- Rare but relevant words get high weight.

Code:

```
from sklearn.feature_extraction.text import TfidfVectorizer


docs = ["I love NLP", "NLP is fun"]
tfidf = TfidfVectorizer()


tfidf_matrix = tfidf.fit_transform(docs)
print("Vocabulary:", tfidf.get_feature_names_out())
print ("TF-IDF Matrix:\n", tfidf_matrix.toarray())
```

1.3 One-Hot Encoding

- Assigns a unique vector with one "hot" (1) element per word.

Example:

Words = [‘I’, ‘love’, ‘NLP’]

Word	**Vector**
I	[1, 0, 0]
love	[0, 1, 0]
NLP	[0, 0, 1]

Code:

```
from sklearn.preprocessing import LabelBinarizer


words = ['I', 'love', 'NLP']
encoder = LabelBinarizer()
one_hot = encoder.fit_transform(words)
print("One-Hot Encodings:\n", one_hot)
```

Part 2: Word Embeddings (Word2Vec, GloVe):

2.1 Word2Vec

- Predicts surrounding words (Skip-gram) or center word (CBOW).
- Learns **dense, continuous vectors** for words.
- Captures **semantic similarity**:

king - man + woman ≈ queen

Code (Using Gensim):

```
from gensim.models import Word2Vec


sentences = [["I", "love", "NLP"], ["NLP", "is", "fun"]]
model = Word2Vec(sentences, vector_size=10, window=2, min_count=1, sg=1)


print ("Vector for 'NLP':\n", model.wv['NLP'])
```

Part 3: Contextual Embeddings (BERT, Sentence Transformers):

3.1 BERT Embeddings

- Generates context-aware embeddings.
- "Apple" in "Apple is a fruit" ≠ "Apple released a new phone"
- Used heavily in Gen AI models.

Code (Using Transformers Library):

```
pip install transformers
```

```
from transformers import BertTokenizer, BertModel
import torch

# Load pre-trained BERT
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

sentence = "NLP with Gen AI is powerful"
tokens = tokenizer(sentence, return_tensors='pt')
with torch.no_grad():
    outputs = model(**tokens)

embedding = outputs.last_hidden_state  # shape: (1, tokens, 768)
print ("BERT Embedding shape:", embedding.shape)
```

3.2 Sentence Embeddings (SBERT)

- Generates a **single vector per sentence**, capturing full meaning.
- Great for tasks like semantic search, similarity, retrieval.

Code (Using sentence-transformers):

```
pip install sentence-transformers
```

```
from sentence_transformers import SentenceTransformer


model = SentenceTransformer('all-MiniLM-L6-v2')
sentences = ["I love NLP", "NLP is fun"]


embeddings = model.encode(sentences)
print("Sentence Embeddings:\n", embeddings)
```

Summary Chart

Technique	Captures Context	Dimensionality	Best For
One-Hot	NO	Sparse	Simple word-based models
BoW	NO	Sparse	Basic NLP pipelines
TF-IDF	NO	Sparse	Document classification/search
Word2Vec	No	Dense (100–300)	Word similarity, embeddings
BERT	Yes	Dense (768+)	Gen AI, Transformers
SBERT	Yes	Dense (384–768)	Sentence similarity, retrieval