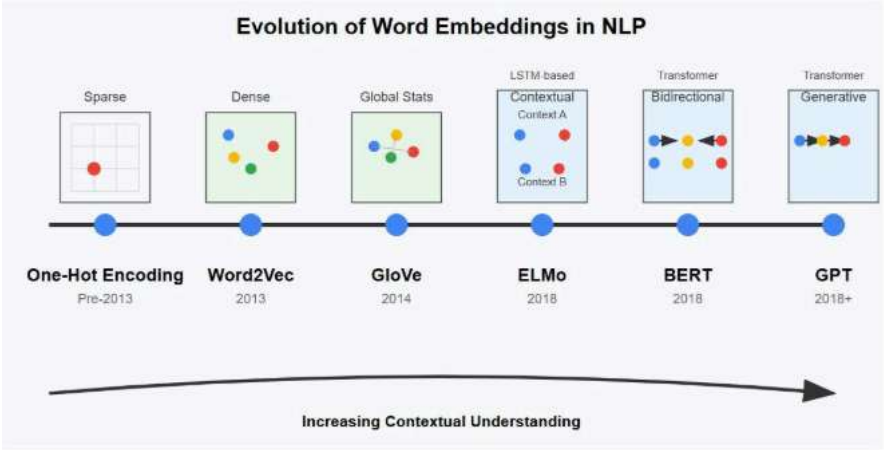
Introduction to LLMs and GenAI

Mini Project 3: Attention Mechanism and Transformers

Emergence of Embeddings from BOW to Self Attention:



Visual Roadmap

```
BOW / TF-IDF (counts)
   ↓
Word2Vec / GloVe (static meaning)
   ↓
BERT / RoBERTa / GPT (contextual meaning)
   ↓
Sentence-BERT / MiniLM (sentence/document embeddings)
   ↓
GPT-3 / GPT-4 / LLaMA (LLMs - reasoning + generation)
   ↓
ChatGPT, Claude, Gemini (GenAI assistants)
```

Quick Recap

- Project 1 (BOW/TF-IDF + ML models): Words treated as independent counts, no meaning.
- Project 2 (Word2Vec, GloVe): Words as static vectors → captures similarity, but same word = same vector everywhere.

Problem still unsolved: Meaning of a word changes with context.

==========

Evolution of Embeddings

1️⃣ Word Embeddings (Word2Vec, GloVe, FastText)

- Each word is mapped to a fixed vector (e.g., 300 dimensions).
- Words with similar context appear close in vector space.

  Example: king − man + woman ≈ queen.
- Limitation:
  - They are static ⟶ the word has only one meaning.

    Example: "bank" (riverbank vs. financial bank) gets the same vector.
  - No understanding of sentence-level meaning.

2️⃣ Contextual Word Embeddings (BERT -Bidirectional Encoder Representations from Transformers, RoBERTa, GPT)

- Instead of static vectors, context matters.

"bank" in "money at the bank" vs. "sat by the river bank" ⟶ different embeddings.

- Each token (word/subword) has an embedding that changes depending on surrounding words.
- Limitation:
  - Great for NLP tasks (QA, classification), but not efficient for sentence-to-sentence similarity.
  - To compare two sentences, BERT must process them together ⟶ very slow for large datasets.

3️⃣ Sentence Embeddings (SBERT = Sentence-Transformers)

- SBERT builds on BERT, but fine-tunes it with Siamese / Triplet networks.

  Output: a single fixed-length vector for the whole sentence/document.
- Now, we can:
  - Compare sentences with cosine similarity (fast).
  - Cluster thousands of sentences/documents easily.

Visual Summary

Level	Example Tech	What It Represents	Limitations
Word Embeddings	Word2Vec, GloVe	Fixed vector for each word	No context, can't represent full sentences
Contextual Word Embeddings	BERT, RoBERTa	Context-sensitive word vectors	Expensive to compare sentences, not sentence-level
Sentence Embeddings	SBERT (MiniLM, MPNet)	One vector per sentence/document	✅ Fast, ✅ Contextual, ✅ Good for clustering & retrieval

Which real-world tasks each technique is best at:

- TF-IDF: Spam detection, keyword extraction
- Word2Vec/GloVe: Synonym finding, word similarity
- BERT: Sentiment analysis, entity recognition, Q&A
- Sentence-BERT/MiniLM: Document classification, semantic search
- GPT/LLMs: Summarization, chatbot, essay writing, code generation

==========

Transformers as the Foundation

The Transformer architecture (Vaswani et al., 2017 — **"Attention Is All You Need"**) is the base building block.

It introduced self-attention, which is very powerful for capturing relationships in sequences (words, tokens, etc.).

What happened after Transformers

Since then, researchers have been:

- Adapting the Transformer for different problems.

- Mixing in older techniques (like Siamese networks, triplet loss, contrastive learning, distillation).
- Fine-tuning them on different datasets (news, dialogues, code, multimodal).

It looks a lot like "hit & trial," but it's actually a structured exploration of design choices:

- Want word embeddings with context? → Use BERT.
- Need faster, smaller models? → DistilBERT, MiniLM.
- Want sentence-level similarity? → SBERT (uses Siamese/Triplet on top of BERT).
- Want long document handling? → Longformer, BigBird.
- Want images + text together? → CLIP.
- Want generation? → GPT family, T5.

Why it feels like "Hit & Trial"

There isn't a single recipe that works for all NLP problems.

Researchers try different training objectives, architectures, and tricks → then benchmark them.

Successful ones become new standards (like BERT, GPT, SBERT, CLIP).

Others fade away if they don't generalize well.

Analogy

- Think of the Transformer as "the engine".
- Some people build cars (BERT, GPT for text).
- Others build bikes (DistilBERT: lightweight).
- Some make trucks (Longformer: long documents).
- Others make flying cars (CLIP: vision + text).

Transformer is the base. Everything else is creative variations + smart training tricks to solve specific tasks.

==========

Problem Statement

Business Context

In today's fast-paced media industry, swiftly categorizing and curating content is crucial. With an overwhelming flow of news across diverse topics, efficient systems are needed to deliver the right content to the right audience and maintain engagement.

Key Challenges:

- Information Overload: The vast number of articles makes manual categorization impractical.
- Timeliness: Delays in classification can lead to outdated or misplaced content.

Problem Definition

E-news Express, a news aggregation startup, struggles with efficiently categorizing diverse articles across sports, entertainment, politics, and more. Manual classification is time-consuming, error-prone, and risks delays or reputational damage. To address this, the startup aims to adopt machine learning to automate categorization.

As a data scientist, the task is to build an unsupervised model that groups articles by content and validate results against human labels, ensuring faster, accurate, and personalized news delivery.

Data Dictionary

- **Text**: The main body of the news article

Installing and importing the necessary libraries

```
# installing the sentence-transformers library
!pip install -U sentence-transformers -q
```

```
# to read and manipulate the data
import pandas as pd
import numpy as np
pd.set_option('max_colwidth', None)    # setting column to the maximum column width as per the data

# to visualise data
import matplotlib.pyplot as plt
import seaborn as sns

# to compute distances
from scipy.spatial.distance import cdist, pdist
from sklearn.metrics import silhouette_score

# importing the PyTorch Deep Learning library
import torch

# to import the model
from sentence_transformers import SentenceTransformer

# to cluster the data
from sklearn.cluster import KMeans

# to compute metrics
from sklearn.metrics import classification_report

# to avoid displaying unnecessary warnings
import warnings
warnings.filterwarnings("ignore")
```

Loading the dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
reviews = pd.read_csv("/content/drive/MyDrive/Ashwani/news_articles.csv")
```

```
# creating a copy of the dataset
data = reviews.copy()
```

Data Overview

Checking the first five rows of the data

```
# Print first 5 rows of data
data.head()
```

```
# checking a news article
data.loc[3, 'Text']
```

Checking the last five rows of the data

```
# Print last 5 rows of data
data.tail()
```

Checking the shape of the dataset

```
# print shape of data
data.shape
```

- The data comprises of ~2.2k news articles

Checking for missing values

```
# Check for missing values
data.isnull().sum()
```

- There are no mising values in the data

Checking for duplicate values

```
# Check for duplicate values
data.duplicated().sum()
```

- We'll drop the duplicate values in the data.

```
data = data.drop_duplicates()

# resetting the dataframe index
data.reset_index(drop=True, inplace=True)
```

```
data.duplicated().sum()
```

```
data.shape
```

- There are no duplicate values in the data now.

Model Building

What is required?

- We don't want to compare words → we want to categorize full news articles.
- SBERT gives us compact, meaningful vectors for each article.
- Then we can use clustering / classification to group them into Politics, Business, Entertainment, Technology, Sports.

What are Sentence-Transformers (SBERT)?

- Sentence-Transformers (SBERT) is an open-source Python framework (built on PyTorch & Hugging Face Transformers) that provides models to generate dense vector representations (embeddings) for sentences, paragraphs, or documents.
- These embeddings capture semantic meaning — so sentences with similar meaning are close together in vector space.

Defining the model

We'll be using the **all-MiniLM-L6-v2** model here.

This model is from Sentence-Transformers (by Hugging Face + Microsoft)

- The all-MiniLM-L6-v2 model is an all-round (all) model trained on a large and diverse dataset of over 1 billion training samples and generates state-of-the-art sentence embeddings of 384 dimensions.
- It is a language model (LM) that has 6 transformer encoder layers (L6) and is a smaller model (Mini) trained to mimic the performance of a larger model (BERT).

- Potential use-cases include text classification, sentiment analysis, and semantic search.

Key reasons it fits our use case well:

- Compact & Lightweight Only 22M parameters, embedding size 384.
- Super fast to train/infer, even on CPU or a free Colab GPU. Ideal for a dataset of ~2K articles — no overkill.
- High-Quality Semantic Embeddings Trained on 1 billion+ sentence pairs.
- Optimized for semantic similarity & clustering, which is exactly what we need for news categorization (unsupervised or semi-supervised).
- Balanced Accuracy vs. Speed Benchmarks show 80–85% performance of much larger models (like MPNet or RoBERTa-large) at a fraction of the compute cost.
- Perfect when our dataset isn't huge but still needs good accuracy.
- Freely Downloadable & Offline Usable Unlike API-only models (e.g., OpenAI's text-embedding-ada-002), we can fully download and run it in Colab without internet after setup.
- Zero cost + no data privacy concerns.

Other comparable models;

Model	Dimensionality	Size	Speed (Colab)	Accuracy (Semantic Tasks)	Cost / Availability
all-MiniLM-L6-v2 (SBERT)	384	~22M params	🚀 Very Fast (CPU/GPU)	⭐⭐☆ (Good, ~80% of SOTA)	Free, Downloadable (Hugg
all-mpnet-base-v2 (SBERT)	768	~109M params	⚡ Medium (needs GPU)	⭐⭐⭐ (High, ~90% of SOTA)	Free, Downloadable (Hugg
all-distilroberta-v1 (SBERT)	768	~82M params	⚡ Medium	⭐⭐☆ (Good, better than MiniLM but below MPNet)	Free, Downloadable
multi-qa-MiniLM-L6-cos-v1 (SBERT)	384	~22M params	🚀 Very Fast	⭐⭐☆ (Good, optimized for QA/Retrieval)	Free, Downloadable
OpenAI text-embedding-3-small	1536	API-only	⚡ Fast (API latency ~300ms)	⭐⭐⭐⭐ (SOTA on benchmarks)	Low cost ($0.02 / 1M toker
Cohere embed-multilingual-v3	1024	API-only	⚡ Fast (API latency)	⭐⭐⭐⭐ (Strong multilingual)	Free tier (5M tokens/mo)
Google Gecko (Vertex AI)	768–1024	API-only	⚡ Fast	⭐⭐⭐⭐ (Google quality, multilingual)	Paid (GCP pricing)

==========

- hf_xet is a helper package for enhancing file transfers with the Hugging Face Hub. It integrates Rust-based code for efficient, chunk-based deduplication, and caching when uploading or downloading large files

```
!pip install hf_xet
```

```
#Defining the model
model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
```

Encoding the dataset

```
# setting the device to GPU if available, else CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
# encoding the dataset
embedding_matrix = model.encode(data['Text'], show_progress_bar=True, device=device)
```

```
# printing the shape of the embedding matrix
embedding_matrix.shape
```

Embedding dimensions for different models

Model	Embedding Dimension
`all-MiniLM-L6-v2`	384
`all-mpnet-base-v2`	768
`paraphrase-MiniLM-L12-v2`	384
`distilbert-base-nli-stsb-mean-tokens`	768

- Each news article has been converted to a 384-dimensional vector

```
# printing the embedding vector of the first review in the dataset
embedding_matrix
```

Note: We have not trained or fine-tuned the model. We have used the pre-trained model to encode the dataset.

Semantic Search

```
# defining a function to compute the cosine similarity between two embedding vectors
def cosine_score(text1,text2):
    # encoding the text
    embeddings1 = model.encode(text1)
    embeddings2 = model.encode(text2)

    # calculating the L2 norm of the embedding vector
    norm1 = np.linalg.norm(embeddings1)
    norm2 = np.linalg.norm(embeddings2)

    # computing the cosine similarity
    cosine_similarity_score = ((np.dot(embeddings1,embeddings2))/(norm1*norm2))

    return cosine_similarity_score
```

```
a= "i love apple"
b= "apple is a fruit"
c= "i like this table"
print(cosine_score(a,b))
print(cosine_score(b,c))
print(cosine_score(a,c))
```

```
# We can also use prebuilt method to calculate similarity score

a= "i love apple"
b= "apple is a fruit"
c= "i like this table"

from sentence_transformers import util

embeddings1 = model.encode(a)
embeddings2 = model.encode(b)
embeddings3 = model.encode(c)

print(util.cos_sim(embeddings1, embeddings2))
print(util.cos_sim(embeddings2, embeddings3))
print(util.cos_sim(embeddings1, embeddings3))
```

Now, let's search for similar reviews in our dataset.

```
# defining a function to find the top k similar sentences for a given query
def top_k_similar_sentences(embedding_matrix, query_text, k):
    # encoding the query text
    query_embedding = model.encode(query_text)

    # calculating the cosine similarity between the query vector and all other encoded vectors of our dataset
    score_vector = np.dot(embedding_matrix,query_embedding)

    # sorting the scores in descending order and choosing the first k
    top_k_indices = np.argsort(score_vector)[::-1][:k]

    # returning the corresponding reviews
    return data.loc[list(top_k_indices), 'Text']
```

```
# defining the query text
query_text = "Budget for elections"

# displaying the top 2 similar sentences
top_k_reviews = top_k_similar_sentences(embedding_matrix, query_text, 2)

for i in top_k_reviews:
    print(i, end="\n")
```

```
    print("**************************************************************")
    print("\n")
```

```
# defining the query text
query_text = "High imports and exports"

# displaying the top 2 similar sentences
top_k_reviews = top_k_similar_sentences(embedding_matrix, query_text, 2)

for i in top_k_reviews:
    print(i, end="\n")
    print("**************************************************************")
    print("\n")
```

Categorization

We'll use K-Means Clustering to categorize the data.

```
meanDistortions = []
clusters = range(2, 11)

for k in clusters:
    clusterer = KMeans(n_clusters=k, random_state=1)
    clusterer.fit(embedding_matrix)

    prediction = clusterer.predict(embedding_matrix)

    distortion = sum(
        np.min(cdist(embedding_matrix, clusterer.cluster_centers_, "euclidean"), axis=1) ** 2
    )
    meanDistortions.append(distortion)

    print("Number of Clusters:", k, "\tAverage Distortion:", distortion)
```

```
plt.plot(clusters, meanDistortions, "bx-")
plt.xlabel("k")
plt.ylabel("Average Distortion")
plt.title("Selecting k with the Elbow Method", fontsize=20)
plt.show()
```

- The appropriate value of k from the Elbow curve seems to 5.

Let's check the silhouette scores.

```
sil_score = []
cluster_list = range(2, 11)

for n_clusters in cluster_list:
    clusterer = KMeans(n_clusters=n_clusters, random_state=1)

    preds = clusterer.fit_predict((embedding_matrix))

    score = silhouette_score(embedding_matrix, preds)
    sil_score.append(score)

    print("For n_clusters = {}, the silhouette score is {})".format(n_clusters, score))
```

```
plt.plot(cluster_list, sil_score, "bx-")
plt.show()
```

- The silhouette coefficient for 5 clusters is the highest.
- So, we will proceed with 5 clusters.

```
# defining the number of clusters/categories
n_categories = 5

# fitting the model
kmeans = KMeans(n_clusters=n_categories, random_state=1).fit(embedding_matrix)
```

```
# checking the cluster centers
centers = kmeans.cluster_centers_
centers
```

```
kmeans.labels_
```

```
# creating a copy of the data
clustered_data = data.copy()

# assigning the cluster/category labels
clustered_data['Category'] = kmeans.labels_

clustered_data.head()
```

```
clustered_data.sample(5)
```

Let's check a few random news articles from each of the categories.

```
# for each cluster, printing the 5 random news articles
for i in range(5):
    print("CLUSTER",i)
    print(clustered_data.loc[clustered_data.Category == i, 'Text'].sample(5, random_state=1).values)
    print("**********************************************************************")
    print("\n")
```

Based on the above news articles, we can see that they can be categorized as follows:

- 0: Sports
- 1: Politics
- 2: Entertainment
- 3: Business
- 4: Technology

```
# dictionary of cluster label to category
category_dict = {
    0: 'Sports',
    1: 'Politics',
    2: 'Entertainment',
    3: 'Business',
    4: 'Technology'
}
# mapping cluster labels to categories
clustered_data['Category'] = clustered_data['Category'].map(category_dict)

clustered_data.head()
```

Comparing with Actual Categories

```
# loading the actual labels
labels = pd.read_csv("/content/drive/MyDrive/Ashwani/news_article_labels.csv")
```

```
labels.shape
```

```
# checking the unique labels
labels['Label'].unique()
```

```
labels['Label'].value_counts(normalize=True)
```

```
# adding the actual categories to our dataframe
clustered_data['Actual_Category'] = labels['Label'].values
```

```
print(classification_report(clustered_data['Actual Category'], clustered_data['Category']))
```

```
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
```

```
cm= confusion_matrix(clustered_data['Actual Category'], clustered_data['Category'],labels= clustered_data['Actual Category'].unique
cm
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=clustered_data['Actual Category'].unique())
disp.plot();
```

```
import matplotlib.pyplot as plt
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=clustered_data['Actual Category'].unique()
)
ax = disp.plot(cmap="Blues").ax_

# Rotate xtick labels by 45°
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha="right")
plt.show()
```

We were able to categorize the news articles with 96% accuracy

- Note that we were able to do so WITHOUT knowing the actual labels for the news articles.

Analyzing Incorrect Predictions

Let's check a few cases where our model incorrectly categorized the news articles.

```
# creating a dataframe of incorrect categorizations
incorrect_category_data = clustered_data[clustered_data['Actual Category'] != clustered_data['Category']].copy()
incorrect_category_data.shape
```

```
incorrect_category_data.head()
```

```
embedding_matrix[24].reshape(1,-1)
```

```
kmeans.cluster_centers_[[2]]
```

```
idx = 24

print('Distance from Actual Category')
print(cdist(embedding_matrix[idx].reshape(1,-1), kmeans.cluster_centers_[[2]], "euclidean")[0,0])

print('Distance from Predicted Category')
print(cdist(embedding_matrix[idx].reshape(1,-1), kmeans.cluster_centers_[[3]], "euclidean")[0,0])
```

We see that the distance of the particular news article from the cluster centers of the actual and predicted categories is almost similar.

```
idx = 45

print('Distance from Actual Category')
print(cdist(embedding_matrix[idx].reshape(1,-1), kmeans.cluster_centers_[[2]], "euclidean")[0,0])

print('Distance from Predicted Category')
print(cdist(embedding_matrix[idx].reshape(1,-1), kmeans.cluster_centers_[[4]], "euclidean")[0,0])
```

We see that the distance of the particular news article from the cluster centers of the actual and predicted categories is almost similar.

Based on this, we can say that a better approach of categorizing these news articles would be to assign more than one category to these news article.

Conclusion

We did the following in the case study:

1. Encoded the dataset using the ***all-MiniLM-L6-v2*** transformer model to generate embeddings of 384 dimensions
2. Queried the dataset to find news articles similar to the query text we passed
3. Categorized the news articles using K-Means Clustering on the transformer encodings
4. Compared the predicted categories of the news articles to the actual categories
5. Analyzed the incorrect predictions to understand where the model went wrong

- Our model can correctly categorize 96% of the news articles.
- As mentioned, one can try tagging news articles with more than one category for better categorization.
    - One can find the cluster centers to which the news article is the closest and assign one or more categories accordingly
- Another approach that can be tried out would be fine-tuning the model to this particular data with category labels (one or more than one) to try and improve the overall performance.
- In addition, the startup can use other transformer models to generate summaries of the news articles, which can provide a gist of the news content.

Thanks!
