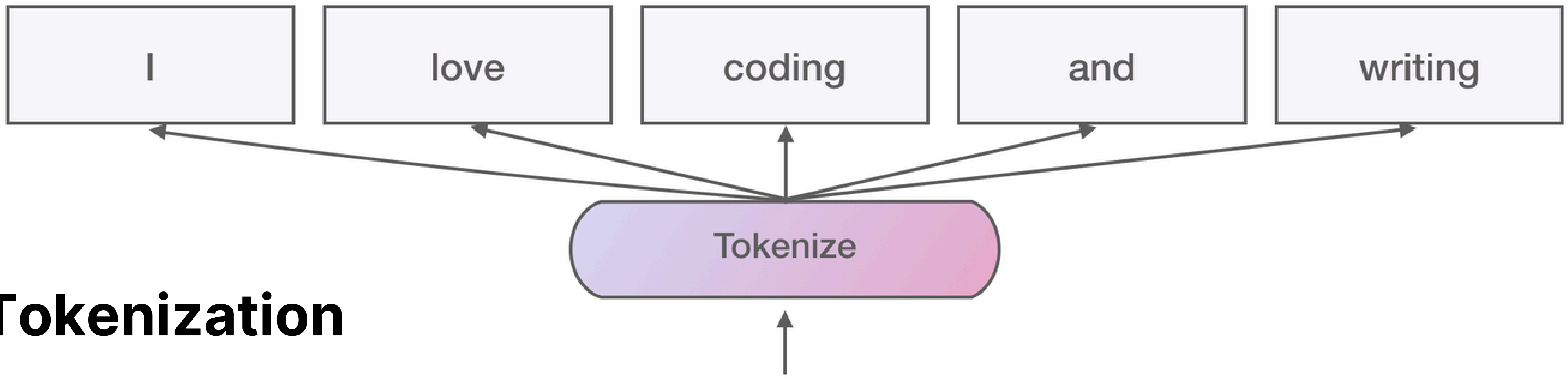
Analytics Vidhya

# Mastering LLMs

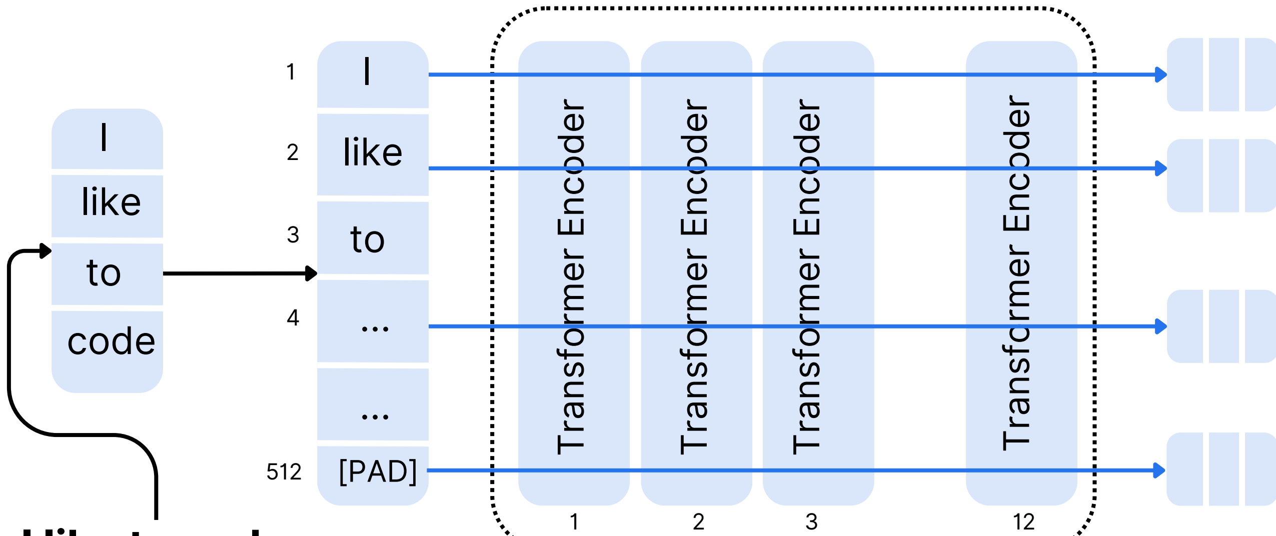
## Day 5: Tokenization and Embeddings

### Tokenization




**“I love coding and writing”**

### Embeddings




**I like to code**

```python
# Importing the necessary library
from transformers import AutoTokenizer

# Load a pre-trained tokenizer (e.g., BERT tokenizer)
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

# Input text to tokenize
text = "Transformers are revolutionizing natural language processing."

# Tokenize the text
tokenized_output = tokenizer(text)

# Print the raw tokenized output
print("Raw Tokenized Output:")
print(tokenized_output)

# Print each component of the tokenized output
print("\nTokens (Text):",
tokenizer.convert_ids_to_tokens(tokenized_output['input_ids']))
print("Input IDs:", tokenized_output['input_ids'])

print("Attention Mask:", tokenized_output['attention_mask'])

# Decode the token IDs back to text
decoded_text = tokenizer.decode(tokenized_output["input_ids"])
print("\nDecoded Text:", decoded_text)

# Additional exploration: Tokenization with padding and truncation
tokenized_with_padding = tokenizer(
    text,
    padding="max_length",  # Adds padding
    max_length=12,         # Ensures the sequence length is at most 12
    truncation=True        # Truncates longer sequences
)

print("\nTokenized with Padding and Truncation:")
print("Input IDs:", tokenized_with_padding["input_ids"])
print("Attention Mask:", tokenized_with_padding["attention_mask"])

# Batch tokenization
batch_texts = ["Transformers are amazing.", "AI is the future."]
batch_tokenized = tokenizer(batch_texts, padding=True, truncation=True)

print("\nBatch Tokenized Output:")
for i, text in enumerate(batch_texts):
    print(f"Text {i + 1}:")
    print("  Input IDs:", batch_tokenized["input_ids"][i])
    print("  Attention Mask:", batch_tokenized["attention_mask"][i])
```

**Tokenization**

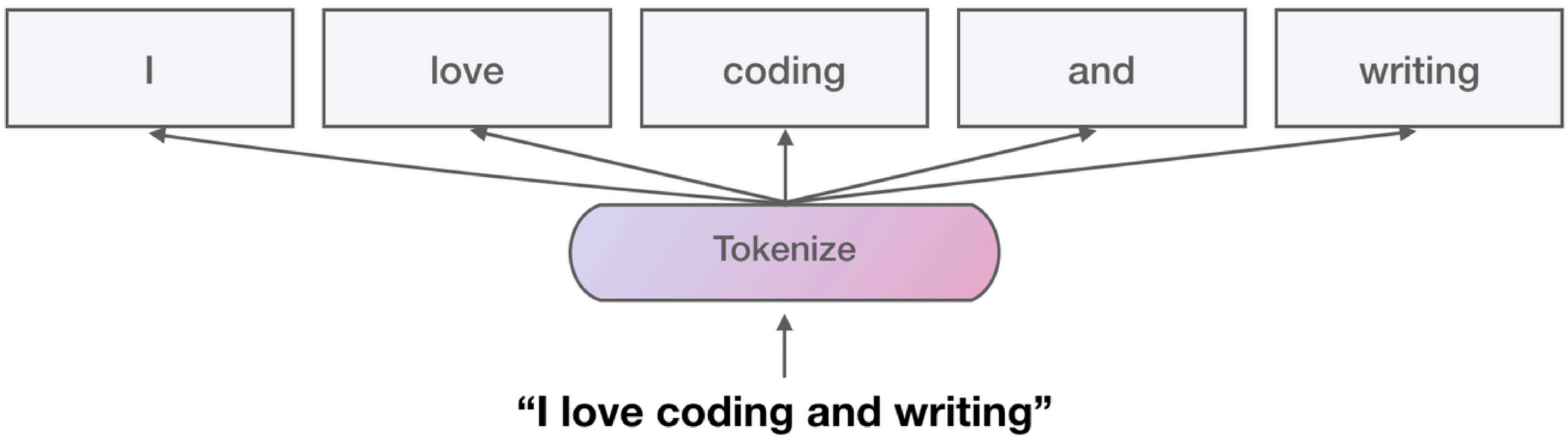
**Embeddings**

**Code to implement**

```python
from transformers import AutoTokenizer, AutoModel
import torch

# Load pre-trained tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = AutoModel.from_pretrained("bert-base-uncased")

# Input text
text = "Transformers are revolutionizing AI and NLP."

# Tokenize the input text
tokenized_output = tokenizer(text, return_tensors="pt", padding=True, truncation=True)

# Extract input IDs and attention mask
input_ids = tokenized_output['input_ids']  # Shape: [batch_size, sequence_length]
attention_mask = tokenized_output['attention_mask']  # Shape: [batch_size, sequence_length]

# Pass the tokenized input through the model to get embeddings
with torch.no_grad():
    model_output = model(input_ids=input_ids, attention_mask=attention_mask)

# Extract the embeddings (last hidden states)
embeddings = model_output.last_hidden_state  # Shape: [batch_size, sequence_length,
embedding_dim]
# Display embedding details
print("Embeddings Shape:", embeddings.shape)
print("First Token Embedding:", embeddings[0][0])  # Embedding for the [CLS] token

# Convert token IDs back to tokens for clarity
tokens = tokenizer.convert_ids_to_tokens(input_ids[0])
print("\nTokens and their embeddings:")
for token, embedding in zip(tokens, embeddings[0]):
    print(f"Token: {token}, Embedding (first 5 dimensions): {embedding[:5]}")
```

# Tokenization

- Tokenization is the process of breaking down text into smaller units, called tokens, that a deep learning model can process.

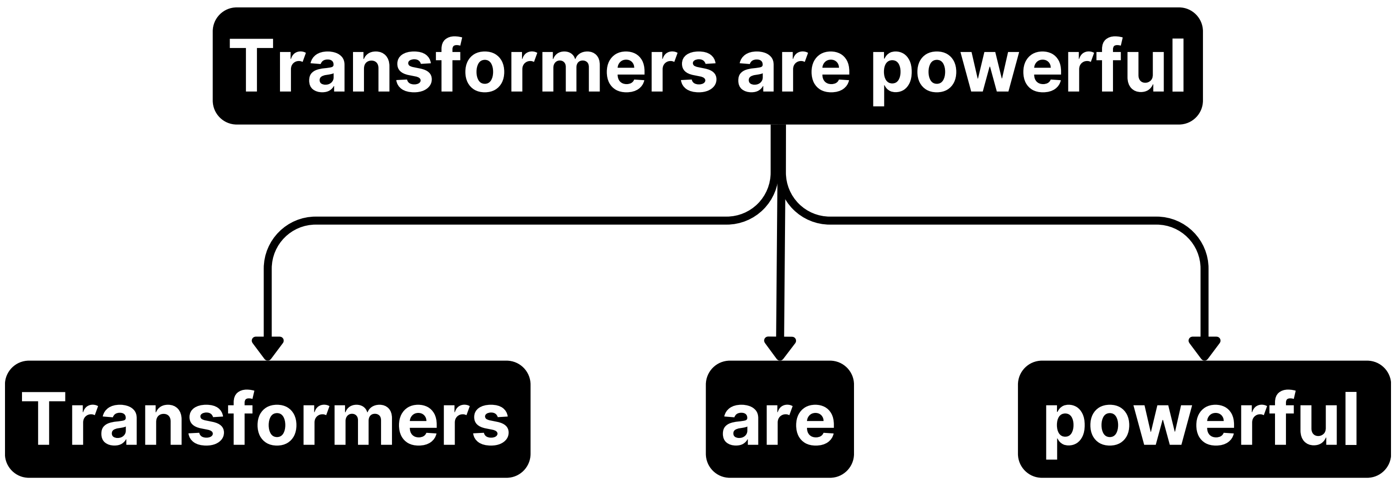



# Types of Tokenization

- Word-based Tokenization
- Subword-based Tokenization
- Character-based Tokenization

# Word-based Tokenization

- Each word is treated as a token.




- **Limitation**: Cannot handle unseen words or word variants effectively (e.g., "transformer" vs. "transformers").

# Subword-based Tokenization

- Breaks words into smaller **meaningful units**, like **prefixes**, **suffixes**, or **roots**.
- Common algorithms:
  - Byte Pair Encoding (BPE)
  - WordPiece
  - SentencePiece
- Example: "transformers" → ["transform", "##ers"] (WordPiece) or ["trans", "former", "s"] (BPE).

# Character-based Tokenization

- Each **character** is treated as a token.
- Example: "Hi" → ["H", "i"]
- Advantage: Handles any input but can result in long sequences.

Here is a Python code to demonstrate the working of **tokenization** in transformers using the **Hugging Face**

```python
# Importing the necessary library
from transformers import AutoTokenizer

# Load a pre-trained tokenizer (e.g., BERT tokenizer)
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

# Input text to tokenize
text = "Transformers are revolutionizing natural language processing."

# Tokenize the text
tokenized_output = tokenizer(text)

# Print the raw tokenized output
print("Raw Tokenized Output:")
print(tokenized_output)

# Print each component of the tokenized output
print("\nTokens (Text):",
tokenizer.convert_ids_to_tokens(tokenized_output['input_ids']))
print("Input IDs:", tokenized_output['input_ids'])

print("Attention Mask:", tokenized_output['attention_mask'])

# Decode the token IDs back to text
decoded_text = tokenizer.decode(tokenized_output["input_ids"])
print("\nDecoded Text:", decoded_text)

# Additional exploration: Tokenization with padding and truncation
tokenized_with_padding = tokenizer(
    text,
    padding="max_length",  # Adds padding
    max_length=12,         # Ensures the sequence length is at most 12
    truncation=True        # Truncates longer sequences
)

print("\nTokenized with Padding and Truncation:")
print("Input IDs:", tokenized_with_padding["input_ids"])
print("Attention Mask:", tokenized_with_padding["attention_mask"])

# Batch tokenization
batch_texts = ["Transformers are amazing.", "AI is the future."]
batch_tokenized = tokenizer(batch_texts, padding=True, truncation=True)

print("\nBatch Tokenized Output:")
for i, text in enumerate(batch_texts):
    print(f"Text {i + 1}:")
    print("  Input IDs:", batch_tokenized["input_ids"][i])
    print("  Attention Mask:", batch_tokenized["attention_mask"][i])
```

Download the **PDF** for a closer look at the code

The explanation of this code is provided on the next slide →

# How This Code Works

**Tokenizer Initialization**:

- A pre-trained tokenizer (bert-base-uncased) is loaded, which matches the vocabulary of the BERT model.

**Tokenization**:

- The input text is tokenized into IDs, and an attention mask is generated.
- Special tokens like [CLS] (start of sequence) and [SEP] (end of sequence) are automatically added.

**Decoding**:

- The tokenized IDs are converted back into readable text to see how the tokenizer represents it.

**Padding and Truncation**:

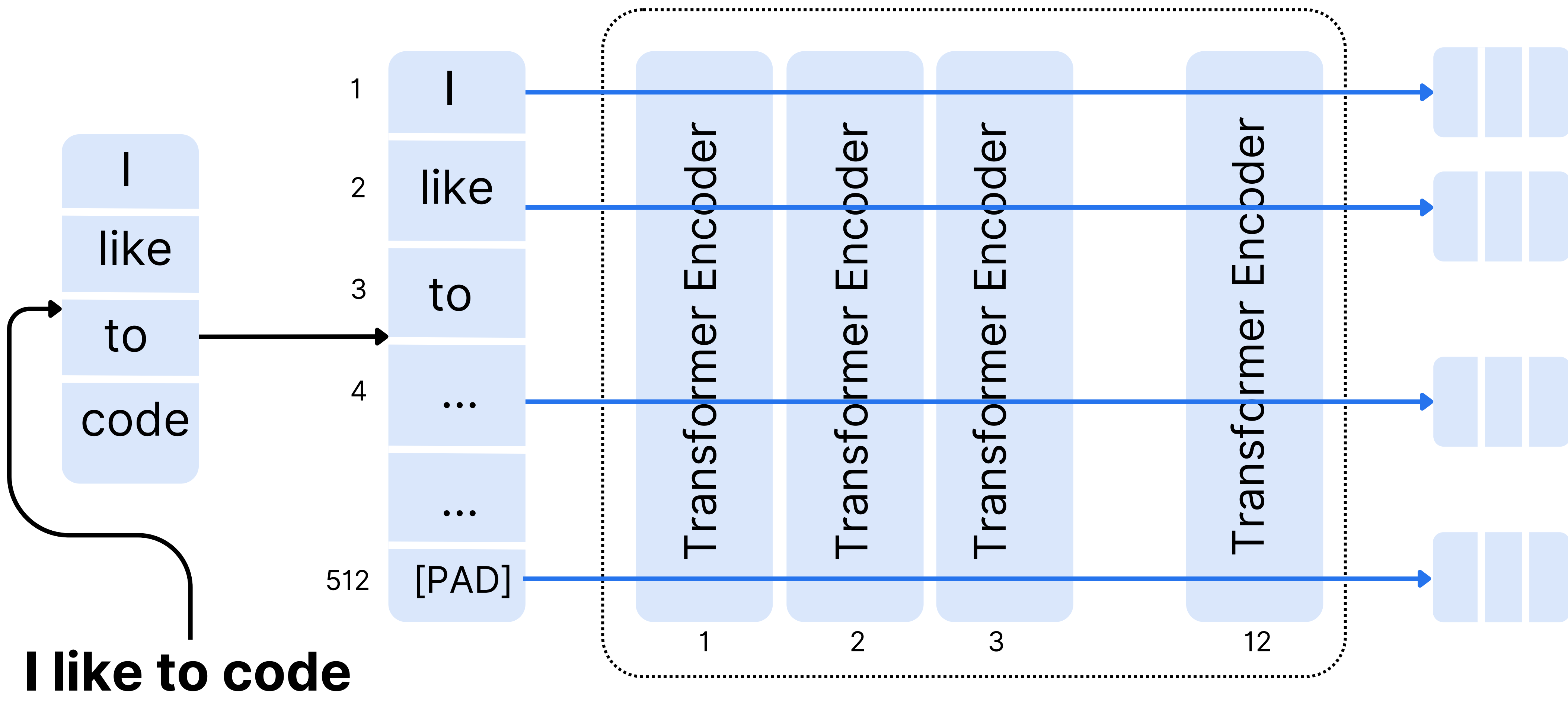
- Ensures that tokenized sequences conform to a fixed length by adding padding tokens or truncating longer texts.

**Batch Tokenization**:

- Tokenizes multiple texts at once, ensuring consistent sequence lengths via padding.

# Embeddings

- Embeddings are dense numerical vector representations of tokens, capturing their meanings, relationships, and context in a lower-dimensional space.




**How It Works:**

- Each token is mapped to a dense vector using an embedding layer (usually implemented as a lookup table).

- The vectors are learned during training to capture semantic relationships.

# Features of Embeddings

## Context-Free Embeddings

- Each word has a fixed representation, regardless of context.
- Example: Word2Vec, GloVe.

## Contextualized Embeddings

- Representations change based on context, thanks to transformer architecture.
- Example: "bank" in "river bank" vs. "financial bank" will have different embeddings.
- Achieved using mechanisms like self-attention.

## Mathematical Representation

- Suppose a vocabulary has V tokens and embedding size is d. The embedding layer is a matrix E of size $V \times d$ times $V \times d$.
- For a token with index i, its embedding vector is E[i].

# Why Important?

Embeddings enable transformers to:

- Capture the semantic similarity between words (e.g., "king" and "queen").
- Represent syntactic and grammatical information.
- Work effectively with numeric data instead of raw text.

# Interaction in Transformers

- **Tokenization** prepares input text, converting it to token IDs.
- **Embedding Layer maps** these IDs to vectors.
- These vectors are passed through the transformer's layers for processing.

# Example Pipeline

Analytics Vidhya

- Input: "I love AI"
- **Tokenization**: [101, 1045, 2293, 2143, 102] (using WordPiece with special tokens)
- **Embedding**: Vectors corresponding to these IDs, ready for transformer layers.

Here is a Python code to demonstrate the working of embedding in transformers:

```
from transformers import AutoTokenizer, AutoModel
import torch

# Load pre-trained tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = AutoModel.from_pretrained("bert-base-uncased")

# Input text
text = "Transformers are revolutionizing AI and NLP."

# Tokenize the input text
tokenized_output = tokenizer(text, return_tensors="pt", padding=True, truncation=True)

# Extract input IDs and attention mask
input_ids = tokenized_output['input_ids']  # Shape: [batch_size, sequence_length]
attention_mask = tokenized_output['attention_mask']  # Shape: [batch_size, sequence_length]

# Pass the tokenized input through the model to get embeddings
with torch.no_grad():
    model_output = model(input_ids=input_ids, attention_mask=attention_mask)

# Extract the embeddings (last hidden states)
embeddings = model_output.last_hidden_state  # Shape: [batch_size, sequence_length,
embedding_dim]
# Display embedding details
print("Embeddings Shape:", embeddings.shape)
print("First Token Embedding:", embeddings[0][0])  # Embedding for the [CLS] token

# Convert token IDs back to tokens for clarity
tokens = tokenizer.convert_ids_to_tokens(input_ids[0])
print("\nTokens and their embeddings:")
for token, embedding in zip(tokens, embeddings[0]):
    print(f"Token: {token}, Embedding (first 5 dimensions): {embedding[:5]}")
```

Download the **PDF** for a closer look at the code

The explanation of this code is provided on the next slide

# How This Code Works

**Tokenizer and Model**:

- **AutoTokenizer**: Loads a tokenizer compatible with bert-base-uncased.
- **AutoModel**: Loads the BERT model to compute embeddings.

**Tokenization**:

- **tokenizer**: Converts input text into input_ids and attention_mask required for the model.
- **padding=True**: Ensures all sequences are padded to the same length.
- **truncation=True**: Truncates sequences that exceed the model's maximum length.

**Model Forward Pass**:

- The tokenized input is passed through the model using model(input_ids, attention_mask).
- The output includes:
  - **last_hidden_state**: Token embeddings for each position in the input sequence.
  - Other optional outputs (not used here).

**Embedding Extraction**:

- last_hidden_state contains embeddings for all tokens in the input sequence, including special tokens like [CLS] and [SEP].

**Token Inspection**:

- Tokens are paired with their corresponding embeddings for inspection.
- Only the first 5 dimensions of each embedding vector are displayed for readability.

Analytics Vidhya

# January 2025

| Sunday | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday |
|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 |
| 5 | 6 ✓ | 7 ✓ | 8 ✓ | 9 ✓ | 10 ✓ | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 | 31 | |

Stay Tuned for **Day 6** of

# Mastering LLMs