

## Day 11: Word Emebeddings

```
# Import necessary modules from the Hugging Face transformers and PyTorch libraries
from transformers import AutoTokenizer, AutoModel
import torch

# Step 1: Initialize the tokenizer
# Load the pre-trained tokenizer automatically for a given model name
tokenizer = AutoTokenizer.from_pretrained('bert-base-uncased')

# Step 2: Define a sample text to tokenize
sample_text = "Transformers are very powerful models for natural language processing."

# Step 3: Tokenize the input text
# Convert text to input tensors required for the model
inputs = tokenizer(sample_text, return_tensors="pt", padding=True, truncation=True)

# Step 4: Print the tokenized output
print("Tokenized Input IDs:", inputs['input_ids'])
print("Token Type IDs:", inputs['token_type_ids'])
print("Attention Mask:", inputs['attention_mask'])

# Step 5: Initialize the pre-trained model
model = AutoModel.from_pretrained('bert-base-uncased')

# Step 6: Pass the tokenized input to the model to get the embeddings
outputs = model(**inputs)

# Step 7: Extract the last hidden state (word embeddings for each token)
word_embeddings = outputs.last_hidden_state

# Step 8: Print the shape of the embeddings
# Shape explanation: (batch_size, sequence_length, hidden_size)
print("Word Embeddings Shape:", word_embeddings.shape)

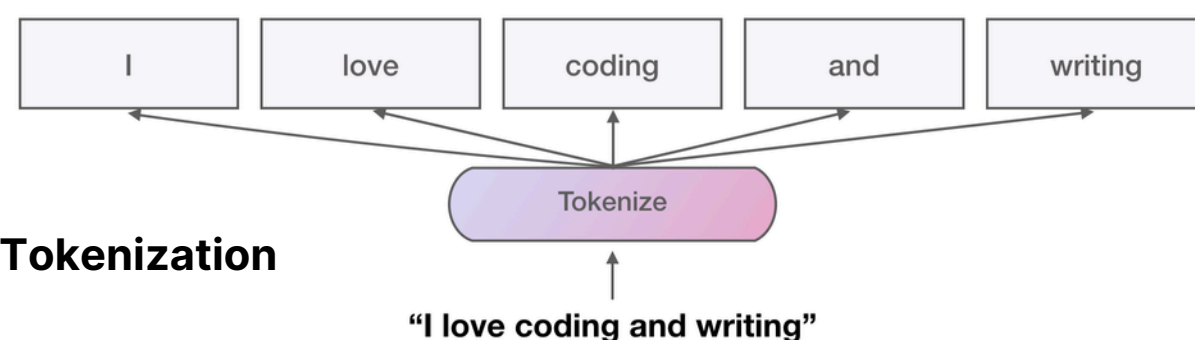
# Step 9: Retrieve embeddings for individual words
# Example: Get the embeddings for the first token (CLS token)
cls_embedding = word_embeddings[0, 0, :] # CLS token at index 0
print("CLS Token Embedding Shape:", cls_embedding.shape)

# Step 10: Convert embeddings to numpy for further processing if needed
cls_embedding_numpy = cls_embedding.detach().numpy()
print("CLS Token Embedding:", cls_embedding_numpy)
```

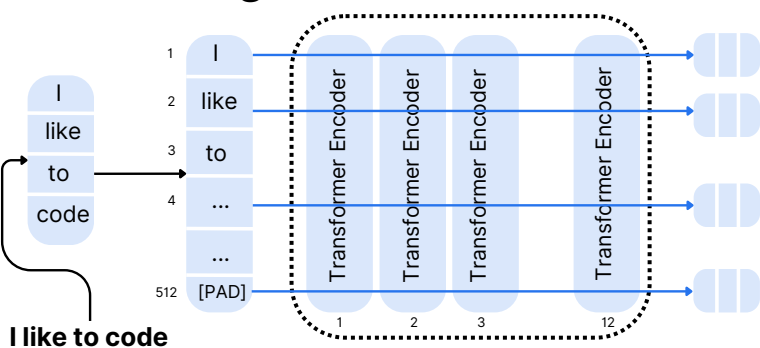
Up to this point, we've only explored the theoretical aspects; now, let's dive into implementing word embeddings in Python. Yesterday, we covered the implementation of tokenization in Python

## Mastering LLMs

### Day 5: Tokenization and Embeddings



### Embeddings



```
# 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 input 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 the token IDs
tokenizer.convert_ids_to_tokens(tokenized_output['input_ids'])
print("Token IDs -> tokenized output['input_ids']")

# Print the attention mask
tokenizer.convert_ids_to_tokens(tokenized_output['attention_mask'])
print("Attention Mask -> tokenized output['attention_mask']")

# Decode the token IDs back to text
decoded_text = tokenizer.decode(tokenized_output['input_ids'])
print("Decoded text -> decoded text")

# Additional visualization: tokenization with padding and truncation
text = "Transformers are revolutionizing natural language processing."
padding = "max_length", # Add padding
max_length=12, # Set the maximum sequence length to at most 12
truncation=True # Truncate longer sequences

tokenized_with_padding = tokenizer(
    text,
    padding=padding,
    max_length=max_length,
    truncation=True
)

print("Tokenized with Padding and Truncation")
print(tokenized_with_padding['input_ids'])
print(tokenized_with_padding['attention_mask'])

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

print("Batch Tokenized Output")
for i, text in enumerate(batch_texts):
    print(f"Text {i+1}: {text}")
    print(f"Token IDs -> batch_tokenized['input_ids'][{i}]")
    print(f"Attention Mask -> batch_tokenized['attention_mask'][{i}]")
```

Tokenization

Embeddings



Code to  
implement

Let's learn how to implement word embeddings in 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 ML."

# 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("Tokens and their embeddings:")
for token, embedding in zip(tokens, embeddings[0]):
    print(f"Token: {token}, Embedding (first 5 dimensions): {embedding[:5]}")
```

# Loading the Pre-trained Tokenizer

---

`AutoTokenizer.from_pretrained('bert-base-uncased')` **loads the tokenizer** for the BERT model.

This tokenizer lowercases text and tokenizes it into subwords based on the model's vocabulary.

## Tokenizing the Sample Text

---

`tokenizer(sample_text, return_tensors="pt", padding=True, truncation=True)` **converts text into token IDs.**

Outputs include:

- **input\_ids**: Numerical token representations.
- **token\_type\_ids**: Token segmentation (used in tasks like QA).
- **attention\_mask**: Marks padded tokens (1 for real tokens, 0 for padding).

# Initializing the Pre-trained Model

---

`AutoModel.from_pretrained('bert-base-uncased')` loads the BERT model for **generating embeddings**.

## Generating Word Embeddings

---

Passing tokenized inputs through the model produces the last hidden state, which contains **contextual embeddings** for each token in the sequence.

## Understanding Word Embeddings Shape

---

`outputs.last_hidden_state` gives embeddings of shape (batch\_size, sequence\_length, hidden\_size).

Example: If `sequence_length=12` and `hidden_size=768`, the shape will be (1, 12, 768).

# Extracting Specific Embeddings

---

[0, 0, :] extracts the [CLS] token, which acts as a representation for the entire sentence.

## Converting to Numpy for Further Use

---

.detach().numpy() is used to convert the PyTorch tensor to a NumPy array for downstream processing.

## Output Example (Sample Execution Output)

---

```
Tokenized Input IDs: tensor([[ 101, 19081,  2024,  2200,  2843,  4274,  2005,  3012,  2653,  3973,
  1012,  102]])
Token Type IDs: tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
Attention Mask: tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
Word Embeddings Shape: torch.Size([1, 12, 768])
CLS Token Embedding Shape: torch.Size([768])
CLS Token Embedding: [-0.2034  0.3182 ... -0.2156]
```



# Key Applications of Word Embeddings

---

- **Sentence Classification:** Using [CLS] embedding for sentiment analysis or topic detection.
- **Named Entity Recognition (NER):** Utilizing token-level embeddings for identifying entities in text.
- **Semantic Similarity:** Comparing embeddings of different texts to measure similarity.
- **Downstream NLP Tasks:** Fine-tuning for tasks like translation, summarization, etc.

Stay Tuned for **Day 12** of

**Mastering LLMs**