Task 30

Mostafa Ahmed

LLMS and Foundational NLP Components

LLM as Vectorizer/Embedder:

1- Explain the core difference between **contextual embeddings** (generated by models like BERT, GPT, etc.) and **static embeddings** (e.g., Word2Vec).

   a. Contextual embeddings gives a word its numerical representation with considering the word's context (words before and after)
   b. Static embeddings don't care about context
   c. Example: bank river, and bank debit
   d. For static embeddings, the word 'bank' gets the same vector representation
   e. For contextual embeddings, the word 'bank' gets two different representations in both statements

2- Name and briefly describe three popular methods for generating sentence/document embeddings using an LLM (e.g., CLS token, mean pooling, or specialized models like Sentence-BERT).

   a. CLS token is taken as the fixed sized representation for the entire input sentence or document. It's used for classification tasks, it's intended capture the summary of the entire input.
   b. Mean pooling is a technique to aggregate the contextualized embeddings of all tokens into a fixed-size vector by calculating the mean of all input tokens. This effectively treats all words equally important and give balanced representation of the text.
   c. Sentence-BERT modifies pre-trained models like BERT and fine tunes them on large datasets and aiming at making snetences that are semantically similar to have close embeddings, while dissimilar ones to have far embeddings

LLM as Tokenizer

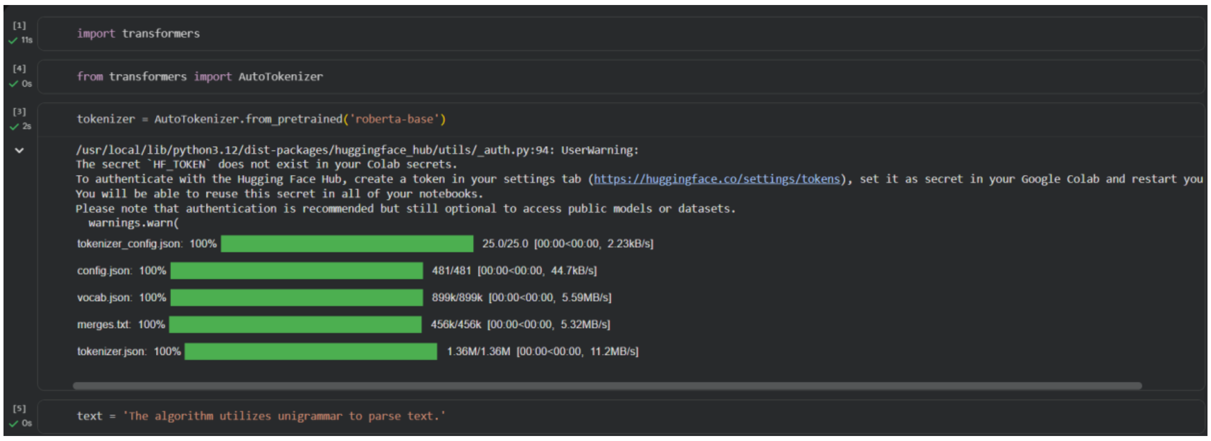
1- Explain the necessity of **subword tokenization** (e.g., Byte-Pair Encoding or WordPiece) in LLMs. How does it handle the **Out-of-Vocabulary (OOV) problem** better than simple word-level tokenization?
   a. Subword tokenization gives a balance between character-level and word-level tokenizations bu keeping the vocabulary size relatively small while maintaining shorter sequence length
   b. The out of vocabulary problem means a model getting a word that was not present during its training, for word-level tokenization, it replaces this word with a generic token
   c. The subword solves this problem by decomposing the unseen word into previously seen subwords

2- List and explain the function of three essential special tokens used by LLM tokenizers (e.g., [CLS], [SEP], [PAD]).
   a. CLS marks the start of a sequence, it's often used as the aggregate representation for the entire input sequence
   b. SEP marks the end of a sequence and is important for separating two distinct sequences in single input.
   c. PAD fills the remaining space of shorter sequences to make sure all inputs has the same length

3- Compare the practical implications of using LLM-derived components:
   a. Trade-offs: Analyze the main trade-offs between using LLM embeddings (e.g., Sentence-BERT or fine-tuned BERT) versus traditional methods (e.g., TF-IDF or GloVe) in terms of accuracy, computational cost, and memory footprint.
      i. For LLM derived embeddings, it has more accuracy, more computational cost and memory footprint
      ii. Traditional methods has less accuracy, less computational cost, and less memory footprint
   b. **Application Domains:** Name one specific real-world application where the superior **semantic richness** of LLM embeddings is a critical advantage (e.g., Semantic Search), and justify your choice.
      i. I would choose product reviews, because LLM embeddings here can extract the semantic of a sentence much more accurately than traditional methods.

TASK 2

2.1:



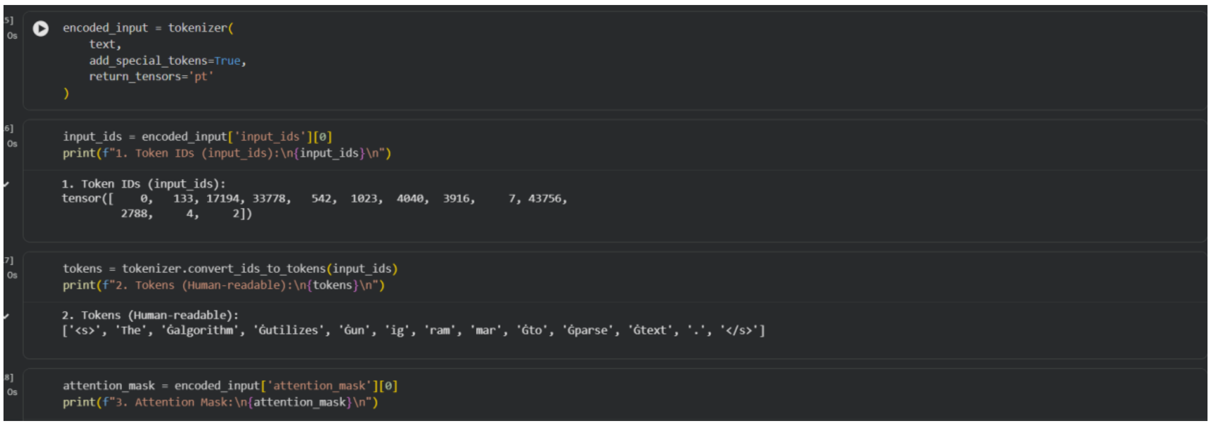
```
import transformers
```

```
from transformers import AutoTokenizer
```

```
tokenizer = AutoTokenizer.from_pretrained('roberta-base')
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart you
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100% 25.0/25.0 [00:00<00:00, 2.23kB/s]
config.json: 100% 481/481 [00:00<00:00, 44.7kB/s]
vocab.json: 100% 899k/899k [00:00<00:00, 5.59MB/s]
merges.txt: 100% 456k/456k [00:00<00:00, 5.32MB/s]
tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 11.2MB/s]
```

```
text = 'The algorithm utilizes unigrammar to parse text.'
```



```
encoded_input = tokenizer(
    text,
    add_special_tokens=True,
    return_tensors='pt'
)
```

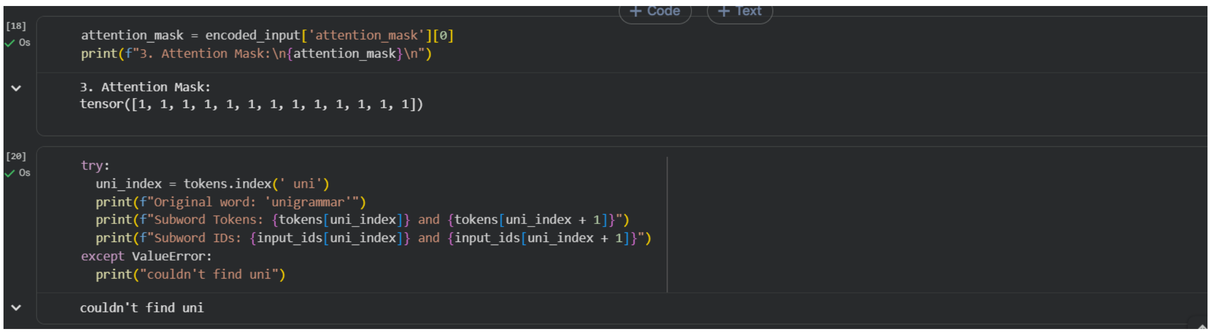
```
input_ids = encoded_input['input_ids'][0]
print(f"1. Token IDs (input_ids):\n{input_ids}\n")
```

```
1. Token IDs (input_ids):
tensor([    0,   133, 17194, 33778,   542,  1023,  4040,  3916,     7, 43756,
         2788,     4,     2])
```

```
tokens = tokenizer.convert_ids_to_tokens(input_ids)
print(f"2. Tokens (Human-readable):\n{tokens}\n")
```

```
2. Tokens (Human-readable):
['<s>', 'The', 'Ġalgorithm', 'Ġutilizes', 'Ġun', 'ig', 'ram', 'mar', 'Ġto', 'Ġparse', 'Ġtext', '.', '</s>']
```

```
attention_mask = encoded_input['attention_mask'][0]
print(f"3. Attention Mask:\n{attention_mask}\n")
```



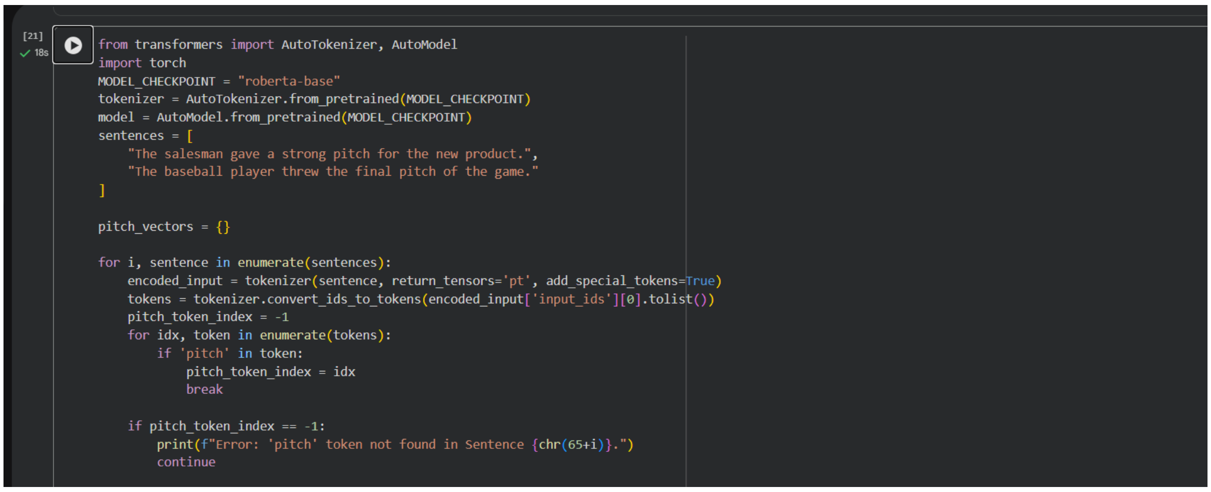
```
attention_mask = encoded_input['attention_mask'][0]
print(f"3. Attention Mask:\n{attention_mask}\n")
```

```
3. Attention Mask:
tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

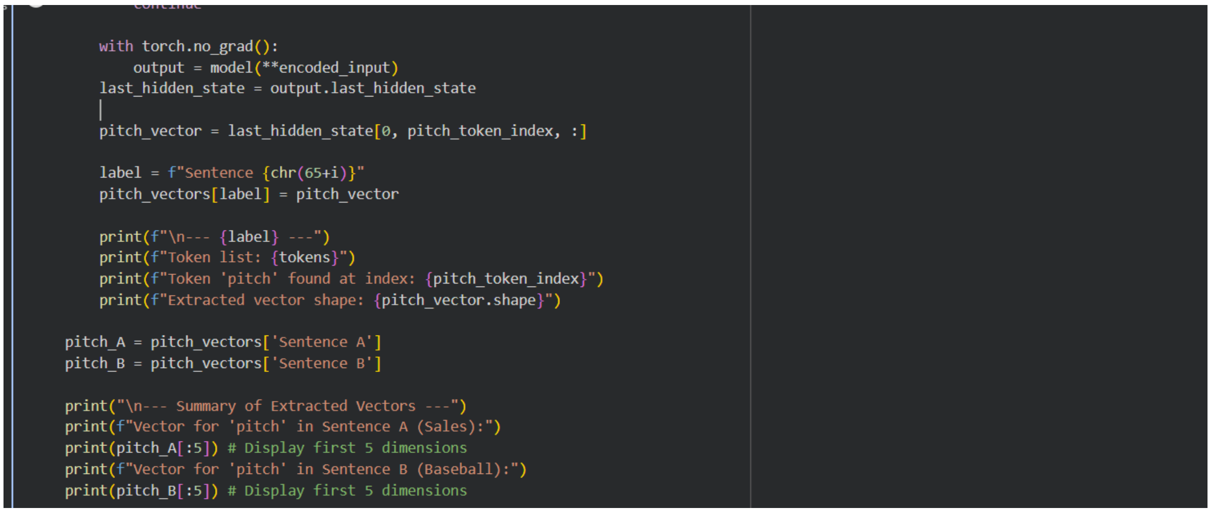
```
try:
  uni_index = tokens.index(' uni')
  print(f"Original word: 'unigrammar'")
  print(f"Subword Tokens: {tokens[uni_index]} and {tokens[uni_index + 1]}")
  print(f"Subword IDs: {input_ids[uni_index]} and {input_ids[uni_index + 1]}")
except ValueError:
  print("couldn't find uni")
```

```
couldn't find uni
```

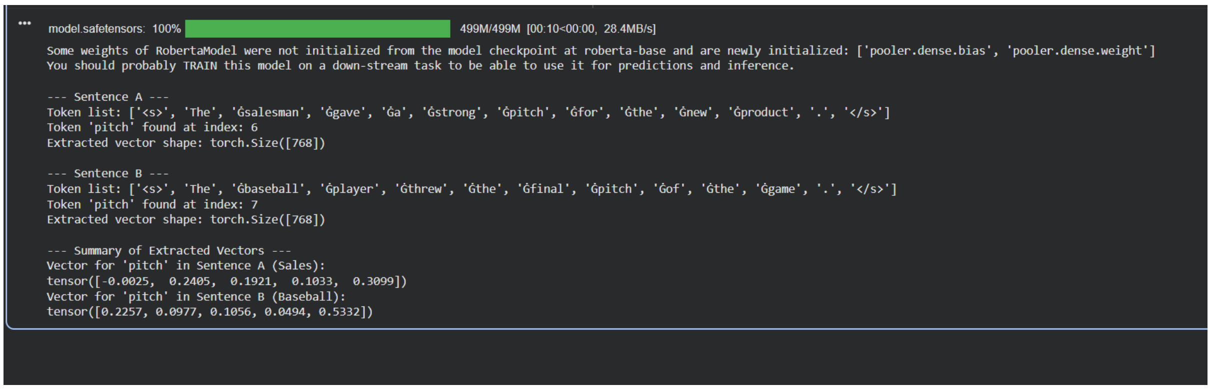
2.2



```
from transformers import AutoTokenizer, AutoModel
import torch
MODEL_CHECKPOINT = "roberta-base"
tokenizer = AutoTokenizer.from_pretrained(MODEL_CHECKPOINT)
model = AutoModel.from_pretrained(MODEL_CHECKPOINT)
sentences = [
    "The salesman gave a strong pitch for the new product.",
    "The baseball player threw the final pitch of the game."
]

pitch_vectors = {}

for i, sentence in enumerate(sentences):
    encoded_input = tokenizer(sentence, return_tensors='pt', add_special_tokens=True)
    tokens = tokenizer.convert_ids_to_tokens(encoded_input['input_ids'][0].tolist())
    pitch_token_index = -1
    for idx, token in enumerate(tokens):
        if 'pitch' in token:
            pitch_token_index = idx
            break

    if pitch_token_index == -1:
        print(f"Error: 'pitch' token not found in Sentence {chr(65+i)}.")
        continue
```



```
            continue

        with torch.no_grad():
            output = model(**encoded_input)
        last_hidden_state = output.last_hidden_state

        pitch_vector = last_hidden_state[0, pitch_token_index, :]

        label = f"Sentence {chr(65+i)}"
        pitch_vectors[label] = pitch_vector

        print(f"\n--- {label} ---")
        print(f"Token list: {tokens}")
        print(f"Token 'pitch' found at index: {pitch_token_index}")
        print(f"Extracted vector shape: {pitch_vector.shape}")

    pitch_A = pitch_vectors['Sentence A']
    pitch_B = pitch_vectors['Sentence B']

    print("\n--- Summary of Extracted Vectors ---")
    print(f"Vector for 'pitch' in Sentence A (Sales):")
    print(pitch_A[:5]) # Display first 5 dimensions
    print(f"Vector for 'pitch' in Sentence B (Baseball):")
    print(pitch_B[:5]) # Display first 5 dimensions
```



```
model.safetensors: 100% 499M/499M [00:10<00:00, 28.4MB/s]
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-base and are newly initialized: ['pooler.dense.bias', 'pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

--- Sentence A ---
Token list: ['<s>', 'The', 'Ġsalesman', 'Ġgave', 'Ġa', 'Ġstrong', 'Ġpitch', 'Ġfor', 'Ġthe', 'Ġnew', 'Ġproduct', '.', '</s>']
Token 'pitch' found at index: 6
Extracted vector shape: torch.Size([768])

--- Sentence B ---
Token list: ['<s>', 'The', 'Ġbaseball', 'Ġplayer', 'Ġthrew', 'Ġthe', 'Ġfinal', 'Ġpitch', 'Ġof', 'Ġthe', 'Ġgame', '.', '</s>']
Token 'pitch' found at index: 7
Extracted vector shape: torch.Size([768])

--- Summary of Extracted Vectors ---
Vector for 'pitch' in Sentence A (Sales):
tensor([-0.0025,  0.2405,  0.1921,  0.1033,  0.3099])
Vector for 'pitch' in Sentence B (Baseball):
tensor([0.2257, 0.0977, 0.1056, 0.0494, 0.5332])
```

Both words are different because the difference in context between two statements

Imagine a model that translates, not sensing the context will make the model extremely weak and unpractical