Hands-On LLM

tokenization : the process of splitting up the sentences into individual words or subwords (*tokens*)

Embeddings : Embeddings are vector representations of data that attempt to capture its meaning.Embeddings are tremendously helpful as they allow us to measure the semantic similarity between two words. Using various distance metrics, we can judge how close one word is to another.if we were to compress these embeddings into a two-dimensional representation, you would notice that words with similar meaning tend to be closer.

There are many types of embeddings, like token embeddings, word embeddings, sentence embeddings, document embeddings that are used to indicate different levels of abstractions (word versus sentence)

Attention allows a model to focus on parts of the input sequence that are relevant to one another ("attend" to each other) and amplify their signal,Attention selectively determines which words are most important in a given sentence.

We can use `transformers` to load both the tokenizer and model. Note that we assume you have an NVIDIA GPU (`device_map="cuda"`)

```
from transformers import AutoModelForCausalLM, AutoTokenize
r

# Load model and tokenizer
model = AutoModelForCausalLM.from_pretrained(
    "microsoft/Phi-3-mini-4k-instruct",
    device_map="cuda",
    torch_dtype="auto",
    trust_remote_code=True,
)
tokenizer = AutoTokenizer.from_pretrained("microsoft/Phi-3-
mini-4k-instruct"
```

Although we now have enough to start generating text, there is a nice trick in transformers that simplifies the process, namely `transformers.pipeline`. It encapsulates the model, tokenizer, and text generation process into a single function:

```
from transformers import pipeline
# Create a pipeline
generator=pipeline("text-generation",
model=model,
tokenizer=tokenizer,
return_full_text=False,
max_new_tokens=500,
do_sample=False
)
```

The following parameters are worth mentioning:

`return_full_text` **By setting this to `False`, the prompt will not be returned but merely the output of the model.**

`max_new_tokens` **The maximum number of tokens the model will generate. By setting a limit, we prevent long and unwieldy output as some models might continue generating output until they reach their context window.**

`do_sample` **Whether the model uses a sampling strategy to choose the next token. By setting this to `False`, the model will always select the next most probable token.**

~~~
# The prompt (user input / query)
messages=[{"role":"user","content":"Create a funny joke abo
ut chickens."}]
# Generate output
output=generator(messages)
print(output[0]["generated_text"])
```
" Why don’t chickens like to go to the gym? Because they ca
n’t crack the egg-sistence of it!"
```
~~~