QUESTION:

model\_ckpt = "google/pegasus-cnn\_dailymail"

tokenizer = AutoTokenizer.from\_pretrained(model\_ckpt)

# Fine tuning data preparation

def convert\_examples\_to\_features(example\_batch):

# Tokenizing the dialogue

input\_encodings = tokenizer(example\_batch['dialogue'] , max\_length = 1024, truncation = True )

# Tokenizing the summary considering them as target

with tokenizer.as\_target\_tokenizer():

target\_encodings = tokenizer(example\_batch['summary'], max\_length = 128, truncation = True )

return {

'input\_ids' : input\_encodings['input\_ids'],

'attention\_mask': input\_encodings['attention\_mask'],

'labels': target\_encodings['input\_ids']

}

print(dataset\_samsum["train"][0]["dialogue"])

Output:

Amanda: I baked cookies. Do you want some?

Jerry: Sure!

Amanda: I'll bring you tomorrow :-)

print(dataset\_samsum["train"][0]["summary"])

Output: Amanda baked cookies and will bring Jerry some tomorrow.

# Testing the function on the first sample from train dataset

tokenized\_example = convert\_examples\_to\_features(dataset\_samsum["train"][0])

print(tokenized\_example["input\_ids"])

Output: [12195, 151, 125, 7091, 3659, 107, 842, 119, 245, 181, 152, 10508, 151, 7435, 147, 12195, 151, 125, 131, 267, 650, 119, 3469, 29344, 1]

print(tokenized\_example["attention\_mask"])

Output:

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

print(tokenized\_example["labels"])

Output:

[12195, 7091, 3659, 111, 138, 650, 10508, 181, 3469, 107, 1]

What is happening in the tokenization process?

ANSWER:

**Explanation of Tokenization**

The code you provided demonstrates tokenization for dialogue summarization using the AutoTokenizer from the Transformers library. Here's a breakdown of what happens:

1. **Loading the tokenizer**:
   * model\_ckpt = "google/pegasus-cnn\_dailymail" defines the pre-trained model checkpoint used for summarization (PEgasus with CNN DailyMail fine-tuning).
   * tokenizer = AutoTokenizer.from\_pretrained(model\_ckpt) loads the tokenizer associated with this model.
2. convert\_examples\_to\_features **function**:
   * This function takes a batch of dialogue-summary examples and prepares them for training.
   * **Dialogue Tokenization**:
     + input\_encodings = tokenizer(example\_batch['dialogue'], max\_length=1024, truncation=True) tokenizes the dialogue text.
       - max\_length=1024 sets a maximum number of tokens allowed for the dialogue.
       - truncation=True ensures the dialogue is cut short if it exceeds the limit.
     + The tokenizer breaks down the dialogue sentence by sentence, likely into words (depending on the model's vocabulary). Each word is replaced with a unique integer ID representing it in the tokenizer's vocabulary.
   * **Summary Tokenization**:
     + with tokenizer.as\_target\_tokenizer(): This context manager switches the tokenizer's behavior to treat the text as a target (summary).
     + target\_encodings = tokenizer(example\_batch['summary'], max\_length=128, truncation=True) tokenizes the summary text similarly to the dialogue, with a shorter maximum length (128 tokens).
   * **Returning the encoded data**:
     + The function returns a dictionary with three key elements:
       - input\_ids: List of integer IDs representing the tokenized dialogue.
       - attention\_mask: A binary mask indicating which tokens are relevant (1) and which are padding (0).
       - labels: List of integer IDs representing the tokenized summary (treated as the labels for training).

**Understanding Attention Mask**

The attention\_mask is crucial for the model to distinguish between actual words and padding tokens. Here's why it's all 1s in this case:

* Truncation is set to True in both dialogue and summary tokenization. This ensures that the input sequences have a fixed length (1024 and 128 respectively).
* Padding tokens are added to shorter sequences to reach the maximum length. These padding tokens are typically assigned a value of 0.
* The attention mask assigns a value of 1 to all tokens in the dialogue and summary because **no padding** was introduced in this specific example. Both sequences reached their maximum length using only actual words from the dialogue and summary.

**Decoding Tokens Back to Text**

The tokenized\_example dictionary contains the encoded dialogue and summary. To get the original sentence back from the tokenized version (list of IDs), you'd need to use the tokenizer's decoding functionality:

Python

decoded\_dialogue = tokenizer.decode(tokenized\_example["input\_ids"], skip\_special\_tokens=True)

decoded\_summary = tokenizer.decode(tokenized\_example["labels"], skip\_special\_tokens=True)

print(f"Decoded Dialogue: {decoded\_dialogue}")

print(f"Decoded Summary: {decoded\_summary}")

This code snippet retrieves the original dialogue and summary sentences from the encoded IDs using the decode method. The skip\_special\_tokens=True argument ensures it excludes any special tokens added by the tokenizer during the process.

QUESTION:  
Can we load a complete trained model from huggingface that does not require fine tuning?

ANSWER:

No, you cannot directly load a complete trained model from Hugging Face that doesn't require any fine-tuning. Here's why:

* **Hugging Face Hub stores pre-trained models:** The models on Hugging Face are primarily pre-trained models. These models are trained on massive datasets for general-purpose tasks like understanding language.
* **Fine-tuning is for specific tasks:** Pre-trained models are powerful baselines, but they need to be adapted to specific tasks through fine-tuning. Fine-tuning involves training the final layers of the model on your own labeled data for your specific problem.

However, there are ways to leverage pre-trained models from Hugging Face with minimal fine-tuning:

* **Transfer Learning:** You can use a pre-trained model for feature extraction and freeze its weights. Then, train a new classifier head on top of the frozen model with your data. This approach utilizes the pre-trained features but adapts to your specific task with minimal fine-tuning.
* **Few-shot Learning:** Some pre-trained models might perform well on your task even with minimal data. This approach requires experimenting with different models to see if acceptable results are achievable without extensive fine-tuning.

Overall, while directly using a model without any fine-tuning isn't possible, Hugging Face offers various pre-trained models and techniques to minimize the fine-tuning required for your specific task.