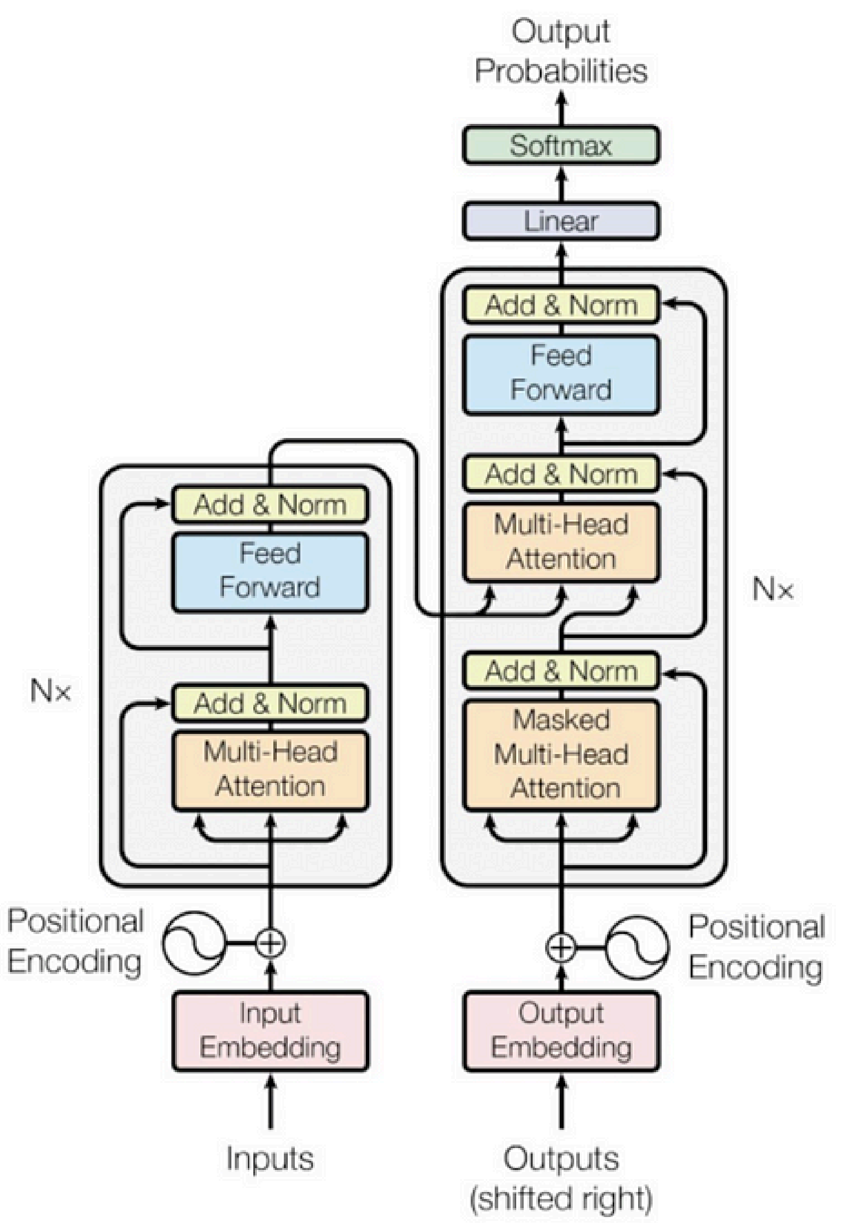
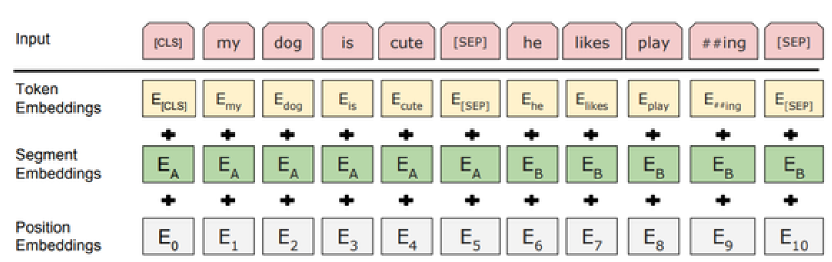
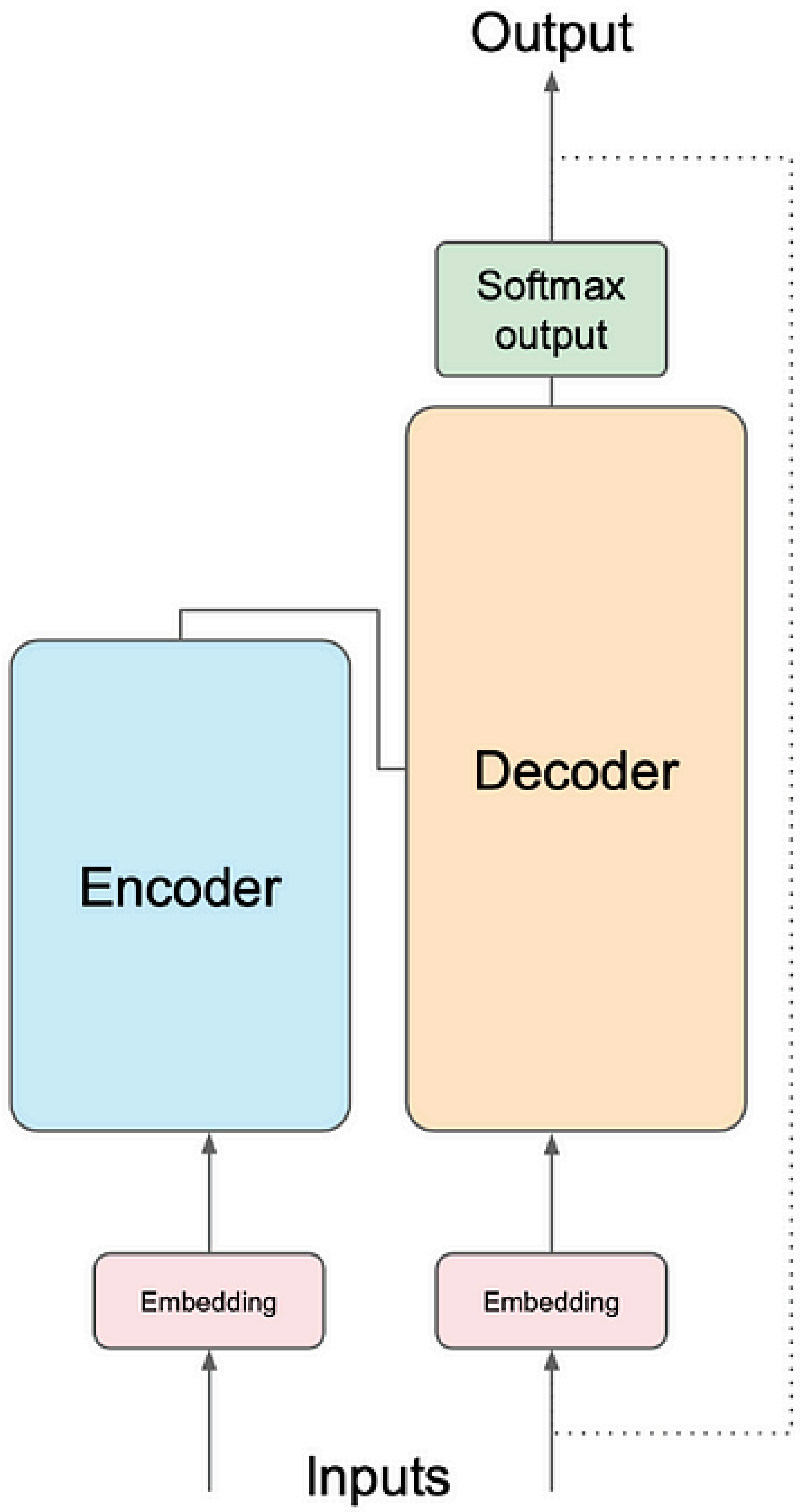
Mastering LLMs

Day 14: BERT(Encoder Model) for Extractive Question and Answering







This post focuses on a practical application of **BERT (Bidirectional Encoder Representations from Transformers)** for **extractive question answering**, where the model identifies a span of text from a given context to answer a question.

```python
from transformers import AutoTokenizer, AutoModelForQuestionAnswering
import torch
import numpy as np

def chunk_context(context, max_length, stride):
    """
    Splits a long context into chunks with overlap (stride).
    """
    tokens = context.split(" ")
    chunks = []
    for i in range(0, len(tokens), max_length - stride):
        chunk = tokens[i:i + max_length]
        chunks.append(" ".join(chunk))
        if len(chunk) < max_length:  # Stop if the last chunk is smaller than max_length
            break
    return chunks

def get_best_answer(model, tokenizer, context, question, max_length=512, stride=128):
    """
    Finds the best answer for a question given a context, handling long contexts with chunking.
    """
    # Tokenize context into manageable chunks
    chunks = chunk_context(context, max_length - 2, stride)  # Reserve space for special tokens
    best_answer = ""
    highest_confidence = 0.0

    for chunk in chunks:
        # Prepare inputs
        inputs = tokenizer.encode_plus(question, chunk, return_tensors="pt", truncation=True)
        input_ids = inputs["input_ids"]
        tokens = tokenizer.convert_ids_to_tokens(input_ids[0])

        # Get model outputs
        with torch.no_grad():
            outputs = model(**inputs)
        start_scores = outputs.start_logits
        end_scores = outputs.end_logits

        # Find the most probable start and end tokens
        start_idx = torch.argmax(start_scores)
        end_idx = torch.argmax(end_scores)

        # Compute confidence
        start_conf = torch.softmax(start_scores, dim=1)[0, start_idx].item()
        end_conf = torch.softmax(end_scores, dim=1)[0, end_idx].item()
        confidence = (start_conf + end_conf) / 2

        # Extract the answer tokens
        if start_idx <= end_idx:
            answer_tokens = tokens[start_idx:end_idx + 1]
            answer = tokenizer.convert_tokens_to_string(answer_tokens)
        else:
            answer = ""

        # Update best answer if confidence is higher
        if confidence > highest_confidence:
            best_answer = answer
            highest_confidence = confidence

    return best_answer, highest_confidence

# Load a pre-trained model and tokenizer
model_name = "bert-large-uncased-whole-word-masking-finetuned-squad"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForQuestionAnswering.from_pretrained(model_name)

# Example usage
if __name__ == "__main__":
    # Define context and question
    context = (
        "The giraffe is an African artiodactyl mammal, the tallest living terrestrial animal, and the
largest ruminant. "
        "Traditionally, giraffes were thought to be one species, Giraffa camelopardalis, with nine
subspecies. However, "
        "recent studies propose that there are eight extant giraffe species."
    )
    question = "How many giraffe species exist?"

    # Get the best answer and confidence score
    answer, confidence = get_best_answer(model, tokenizer, context, question)

    # Print the results
    print(f"Question: {question}")
    print(f"Answer: {answer}")
    print(f"Confidence: {confidence:.2f}")
```

This code handles:

1. **Chunking Long Contexts:** The chunk_context function splits the context into overlapping chunks to handle the token limit.
2. **Extracting Answers:** For each chunk, the model predicts start and end indices, computes confidence scores, and determines the most likely answer.
3. **Confidence Computation:** The confidence is computed as the average probability of the start and end tokens.

Key Steps Covered

1. **Introduction to Extractive Question Answering:**
    - BERT, a transformer-based encoder-only model, is used to extract answers directly from a given context.

2. **Model Selection:**
    - Fine-tuned BERT models for question answering (e.g., trained on the SQuAD dataset) are available on platforms like the **Hugging Face Model Hub.**
    - Reuse pre-trained models when possible to save time and resources.

3. **Preparing Inputs for BERT:**
   - Both the **context** and the **question** are tokenized and concatenated with a special token to create a single input for BERT.
   - BERT predicts **start** and **end indices** of the answer span in the text.

4. **Answer Prediction:**
   - Probabilities for the start and end indices are calculated using **softmax** over logits.
   - The span of text between these indices forms the answer.
   - Example: For the giraffe context, BERT correctly identifies that there are eight species.

5. **Adding Features to the Pipeline:**
   - **Confidence Scores:** Averaging start and end token probabilities to quantify the model's confidence.
   - **Handling Impossible Questions:** Introducing a classification token for questions without an answer in the context (e.g., asking about "dogs" in a giraffe-related context).

6. **Context Length Limitations:**
   - BERT can only process up to **512 tokens**. Longer contexts are truncated, potentially losing relevant information.

- Example: Adding information about coffee production caused the input to exceed the token limit, leading to failed predictions.

7. **Overcoming Length Limitations:**
    - Splitting long contexts into **smaller chunks** using a **sliding window** with overlaps (stride) ensures better coverage of information.
    - Answers are extracted from each chunk, and the one with the highest confidence is selected.

8. **Final Results:**
    - The chunking method successfully handles longer contexts, allowing BERT to process extended information and answer all questions accurately.

Analytics Vidhya

Stay Tuned for **Day 15** of

Mastering LLMs