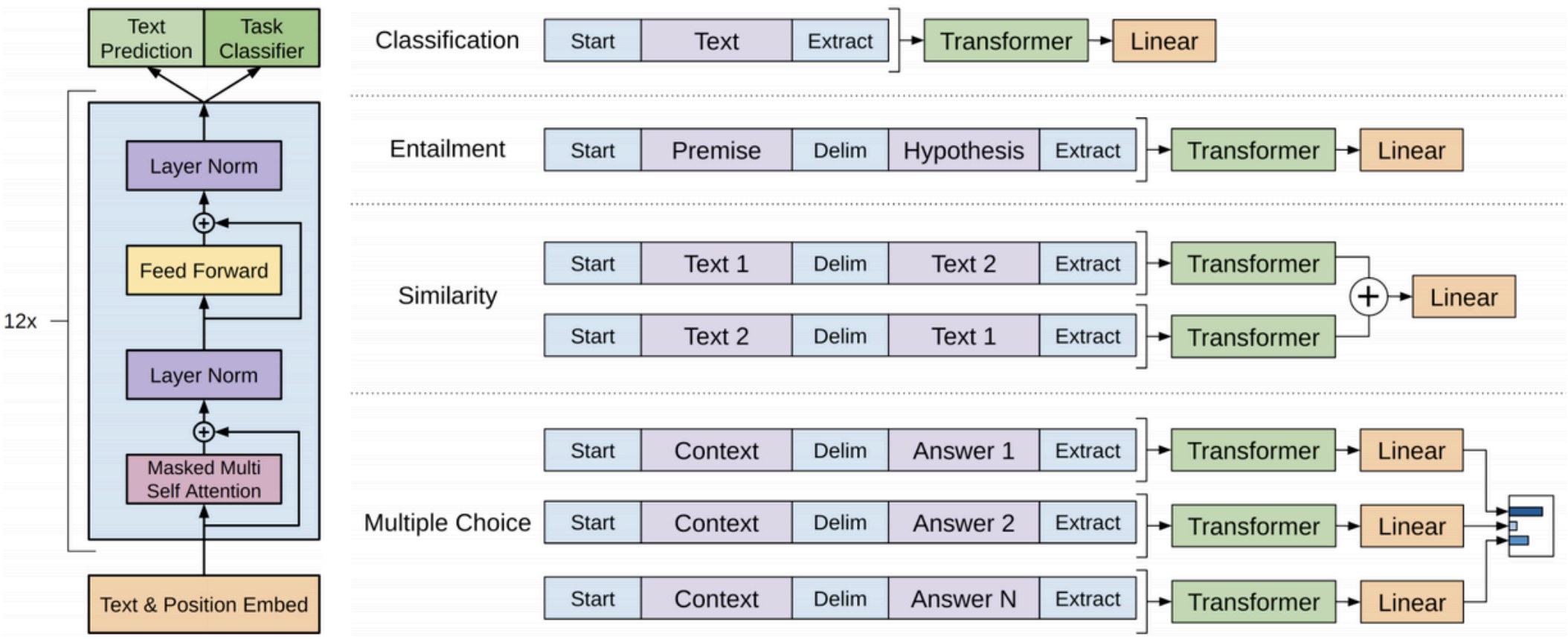
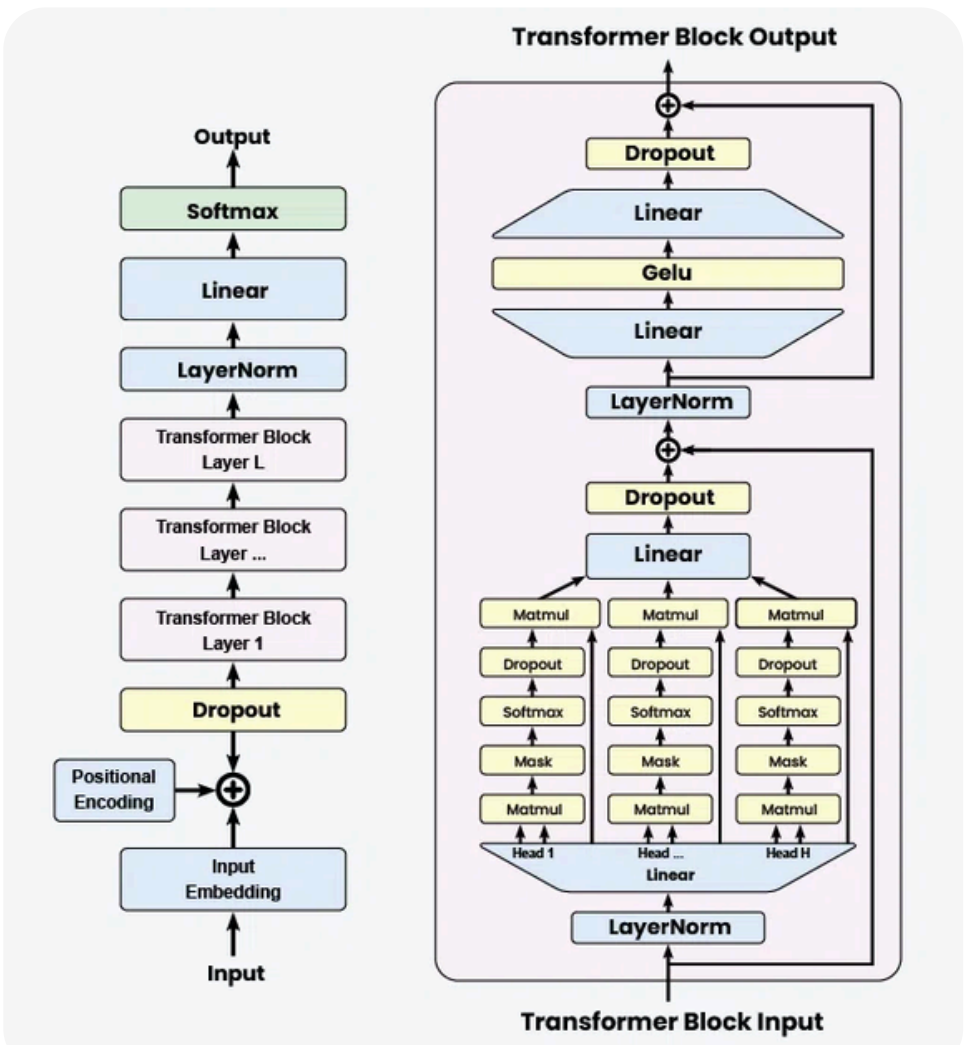
Analytics Vidhya

Mastering LLMs

Day 15: GPT(Decoder Model) for Instruction Following





GPT, short for Generative Pretrained Transformer, is a type of neural network model developed primarily for natural language processing (NLP) tasks.

In this post, the focus is on **fine-tuning a GPT-2 model** to follow instructions, enabling it to perform tasks similar to **ChatGPT**. Here's a concise breakdown of the process:

1. **Objective:**
   - Fine-tune GPT-2, a decoder-only model, to handle instruction-response tasks, similar to conversational and instruction-following models.

2. **Dataset:**
   - Use the **Open Instruct Dataset**, which contains instruction-response pairs (e.g., "Describe a barista's workday" → expected answer).
   - Preprocess the dataset to format it as prompts (instruction + input) and completions (expected output).

3. **Model Preparation:**
   - Use **Diablo GPT**, a GPT-2 variant pre-trained for conversational tasks, as a starting point.
   - Add padding tokens for sequence formatting since GPT-2 lacks a padding mechanism by default.
   - Tokenize and truncate inputs to **120 tokens** for faster processing.

4. **Training Process:**
   - Formulate the task as **causal language modeling**, where the model predicts the next token in a sequence.
   - Use Hugging Face's **Trainer** class to handle the training pipeline.
   - Train on a subset of 100,000 examples to demonstrate the process quickly.

5. **Evaluation:**
   - After training, the fine-tuned model is tested on various tasks, such as:
   - Cooking instructions.
   - Travel recommendations.
   - Financial advice.
   - Problem-solving (e.g., finding the fastest route between cities).
   - The model produces meaningful, albeit basic, responses due to limited training data.

6. **Key Insights:**
   - Fine-tuning enables GPT-2 to follow instructions effectively.
   - With more data and training, the model can achieve more advanced instruction-following capabilities.

Outcome

This lesson demonstrates how to adapt GPT-2 for instruction-following tasks, illustrating the practical steps for fine-tuning a generative language model for specific use cases.

```
from transformers import AutoTokenizer, AutoModelForCausalLM, Trainer, TrainingArguments,
DataCollatorForLanguageModeling
from datasets import load_dataset, Dataset
import torch

def preprocess_data(examples):
    """
    Prepares the input for the model by formatting it as an instruction-response pair.
    """
    prompt = examples["instruction"] + "\n" + examples["input"] + "\n" + "Response: " +
examples["output"]
    return {"text": prompt}

def load_and_prepare_data():
    """
    Loads the dataset and prepares it for training.
    """
    dataset = load_dataset("open-instruct")
    dataset = dataset.map(preprocess_data, remove_columns=["instruction", "input", "output"])
    dataset = dataset.train_test_split(test_size=0.1)
    return dataset

def fine_tune_gpt2():
    """
    Fine-tunes a GPT-2 model on instruction-following tasks.
    """
    model_name = "microsoft/DialoGPT-medium"
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForCausalLM.from_pretrained(model_name)

    # Define a pad token
    tokenizer.pad_token = tokenizer.eos_token

    # Load dataset
    dataset = load_and_prepare_data()
    tokenized_dataset = dataset.map(lambda x: tokenizer(x["text"], truncation=True, max_length=120,
padding="max_length"), batched=True)
```

```
    # Training arguments
    training_args = TrainingArguments(
        output_dir="./gpt2-instruction-following",
        evaluation_strategy="epoch",
        learning_rate=5e-5,
        per_device_train_batch_size=8,
        per_device_eval_batch_size=8,
        num_train_epochs=1,
        weight_decay=0.01,
        save_strategy="epoch",
    )

    # Data collator
    data_collator = DataCollatorForLanguageModeling(
        tokenizer=tokenizer,
        mlm=False  # GPT uses causal language modeling
    )

    # Trainer
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=tokenized_dataset["train"],
        eval_dataset=tokenized_dataset["test"],
        tokenizer=tokenizer,
        data_collator=data_collator,
    )

    # Train the model
    trainer.train()

    # Save model
    model.save_pretrained("./gpt2-instruction-following")
    tokenizer.save_pretrained("./gpt2-instruction-following")

def generate_response(prompt):
    """
    Generates a response from the fine-tuned model given an instruction.
    """
    model_name = "./gpt2-instruction-following"
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForCausalLM.from_pretrained(model_name)

    inputs = tokenizer(prompt, return_tensors="pt", padding=True, truncation=True, max_length=120)
    outputs = model.generate(**inputs, max_new_tokens=50)

    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return response.split("Response:")[-1].strip()

if __name__ == "__main__":
    fine_tune_gpt2()

    # Example test cases
    print(generate_response("Describe the typical workday for a barista"))
    print(generate_response("What's the best way to cook chicken breast?"))
    print(generate_response("Recommend a location for a summer vacation."))
```

Analytics Vidhya

Stay Tuned for **Day 16** of

Mastering LLMs