Week 1: Introduction to Generative AI and Large Language Models (LLMs)

Overview

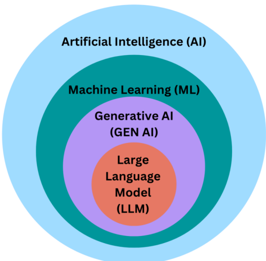
This week, we will introduce you to **Generative AI** and **Large Language Models (LLMs)**. You will learn how LLMs work, their key use cases, and the challenges involved in training them. The topics covered include:

- **Transformer architecture** – the foundation of modern LLMs.
- **Text generation techniques** – how LLMs create human-like text.
- **Memory optimization** – ways to manage computational resources efficiently.
- **Scaling models across GPUs** – how to train and deploy large models.

What is Generative AI?

Generative AI refers to models that create new content, such as text, images, or code, by mimicking human output.

Large Language Models (LLMs) are a subset of generative AI specifically designed for working with text. These models are trained on vast amounts of text data to recognize patterns and generate contextually relevant responses.



Example of How LLMs Work:

When given a sentence starter like **"The sky is..."**, an LLM predicts the next word based on its training data, generating something like **"blue."**

Common Use Cases of LLMs

LLMs can perform various text-based tasks, including:

1. Text Generation

LLMs can write entire articles, essays, or even code.

- **Example:** A chatbot responding with, "How can I assist you today?"

2. Translation

They convert text between languages accurately.

- **Example:** "Hello" → "Hola" (Spanish)

3. Summarization

LLMs can condense long articles into short summaries.

- **Example:** Summarizing a 1,000-word news article into three sentences.

4. Named Entity Recognition (NER)

This identifies important entities (names, dates, places, etc.).

- **Example:** "Steve works at Google" → **Person:** Steve, **Organization:** Google.

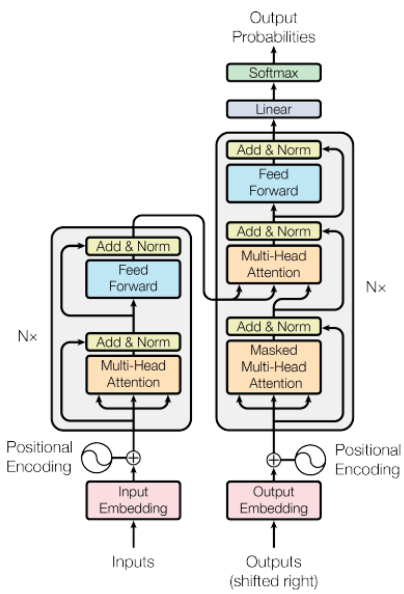
Text Generation Before Transformers

Before **Transformers**, earlier models used **Recurrent Neural Networks (RNNs)**, which processed text sequentially (word-by-word). However, RNNs had limitations:

1. **Short-Term Memory Issues:**
   - RNNs struggled with long sentences.
   - Example: "The cat, which was chased by the dog, ran into the garden." By the time it reached "garden," the model might forget the "cat."
2. **Slow Training Speed:**
   - RNNs processed one word at a time, making them inefficient.

The Transformer Architecture

Introduced in **2017**, **Transformers** revolutionized text generation by allowing models to process all words simultaneously instead of sequentially.



Key Components of a Transformer:

1. **Tokenization:** Converts words into numerical tokens.
   - Example: "Hello" → [23, 45]
2. **Embeddings:** Maps tokens to numerical vectors to represent meaning.
   - Example: "King" → [0.3, -0.2, 0.5]
3. **Positional Encoding:** Ensures word order is considered.
   - Example: "Dog bites man" vs. "Man bites dog" (different meanings)
4. **Self-Attention Mechanism:** Helps understand relationships between words.
   - Example: "The teacher gave the students homework" → "teacher" relates to "students" and "homework."
5. **Feed-Forward Network:** Converts self-attention outputs into probabilities for predicting the next word.

How Transformers Generate Text

When you ask a model to generate a story, the process involves:

1. **Input:** "Write a story about a robot."
2. **Tokenization:** Converts input into numbers like [10, 23, 45].
3. **Processing:** Uses self-attention to predict the next words.
4. **Output:** "Once, a robot named Zeta explored Mars and discovered ancient ruins."

Controlling Output with Parameters

- **Temperature:** Adjusts randomness.
  - **Low (0.1):** Predictable, structured output. Example: "The cat sat on the mat."
  - **High (2.0):** More creative output. Example: "The cat tap-danced on the moon."
- **Max Tokens:** Limits response length. Example: Setting **Max Tokens = 50** ensures the response doesn't exceed ~50 words.

Prompting and Prompt Engineering

Prompting techniques help guide LLM responses effectively:

- **Zero-Shot Prompting:** No examples provided.
  - Example: "Summarize this article: [text]."
- **One-Shot Prompting:** One example provided.
  - Example: "Translate 'Hello' to French: Bonjour. Translate 'Goodbye' to French: ___"
- **Few-Shot Prompting:** Multiple examples provided.
  - Example: "Review: 'Great movie!' → Positive. Review: 'Bad acting' → Negative. Review: [Your text] → ___"

Building a Generative AI Model: Project Lifecycle

1. **Define the Use Case:** Example: "Summarize legal documents."
2. **Choose a Model:** Use an existing model (e.g., T5) or train a custom one.
3. **Adapt the Model:** Fine-tune it with domain-specific data.
4. **Deploy the Model:** Integrate into applications via APIs.

Lab 1: Dialogue Summarization with FLAN-T5

Goal: Summarize customer support conversations.

Steps:

Install necessary libraries:
pip install transformers datasets

1. Load the model:
 from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

   tokenizer = AutoTokenizer.from_pretrained("google/flan-t5-base")

   model = AutoModelForSeq2SeqLM.from_pretrained("google/flan-t5-base")

2. Generate a summary:
 inputs = tokenizer("Summarize: " + conversation_text, return_tensors="pt")

   outputs = model.generate(inputs["input_ids"], max_length=100)

   summary = tokenizer.decode(outputs[0], skip_special_tokens=True)

Challenges in Training LLMs

1. **High GPU Memory Requirements:**
   - A 1B-parameter model requires **24GB** memory (FP32 precision).
2. **Optimization Strategies:**
   - **Quantization:** Reduces memory usage (FP32 → BFLOAT16 → INT8)
   - **Distributed Training:** Splits model across GPUs.
   - Pre-training large language models
     Computational challenges of training LLMs
     Efficient multi-GPU compute strategies
     Scaling laws and compute-optimal models
     Pre-training for Domain Adaptation

Pre-training from scratch is useful when a model needs to specialize in a particular field, such as finance, law, or medicine. In these cases, general-purpose models may not perform well due to a lack of domain-specific vocabulary or context understanding.

When to Train a Model from Scratch

- **Specialized Vocabulary:** Fields like law and medicine use unique terms that general models might not understand well.
  - **Example:** A legal model needs to recognize terms like *res judicata* (a legal principle).
- **Highly Focused Data:** Some industries have proprietary or confidential data that general models lack access to.
  - **Example:** BloombergGPT was trained with 51% financial data to better understand financial reports and market trends.

Benefits of Domain-Specific Pre-Training

- **Enhanced Understanding:** The model gains deeper insights into the domain-specific language.
- **Better Accuracy:** It provides more relevant and precise responses.

- **Improved Performance:** Even smaller models can outperform larger, general models if properly trained on high-quality, domain-specific data.

Computational Challenges of Training LLMs

Key Issues in Training Large Models

1. **GPU Memory Constraints:**
   - Large models require vast amounts of memory.
   - Example: A model with **1 billion parameters** needs **24GB of memory** in FP32 precision.
2. **Optimizer States and Training Overhead:**
   - Advanced optimizers like Adam require **twice the model size** in memory.
   - Example: A **10B-parameter model** needs **80GB** for training (model + optimizer states).
3. **Quantization Techniques for Efficiency:**
   - **FP32 (32-bit floating point):** High precision but memory-intensive.
   - **BFLOAT16 (16-bit floating point):** Reduces memory usage while retaining numerical range.
   - **INT8 (8-bit integer):** Further reduces size but sacrifices some precision.

Why Quantization Works

By using lower precision formats (BFLOAT16, INT8), models can significantly reduce memory usage without a major drop in performance. For example, **BFLOAT16 retains most of FP32's range** but uses **half the memory**, making it a preferred choice for training large-scale LLMs.

Efficient Multi-GPU Compute Strategies

1. Distributed Data Parallel (DDP)

- Copies the entire model onto each GPU.
- Best for **small to medium models** (up to ~2B parameters).
- Example: Training **T5-3B** across 4 GPUs.

2. Fully Sharded Data Parallel (FSDP)

- Splits model parameters, optimizer states, and gradients across multiple GPUs.
- Uses **ZeRO Optimization** to reduce memory footprint.
- **ZeRO Stages:**
  - **Stage 1:** Shards optimizer states (4x memory savings).
  - **Stage 2:** Shards gradients (8x memory savings).
  - **Stage 3:** Shards model parameters (scales to 64+ GPUs).
- Example: Training **T5-11B** across **512 GPUs**.

Trade-offs:

- **DDP** is simpler but uses more memory.
- **FSDP** saves memory but increases GPU communication overhead.

Scaling Laws and Compute-Optimal Models

Findings from the Chinchilla Paper

Research suggests that model performance **depends not just on size but on the amount of training data**.

- **Key Insight:** A model should be trained with **20 times more tokens than its number of parameters** for optimal performance.
  - **Example:** A **70B-parameter model** should be trained on **1.4 trillion tokens**.
- **Smaller, well-trained models can outperform larger ones** with the right amount of data.
  - **Example: Chinchilla-70B** outperforms **GPT-3-175B** because it was trained more efficiently.

Practical Implications

- Instead of making models **bigger**, it is often better to train them **smarter** with **better data**.
- Many companies are shifting toward **high-quality, well-curated datasets** rather than just increasing model size.

  Pre-training for domain adaptation this is missing

Key Takeaways

- **Transformers enable efficient and high-quality text generation.**
- **LLMs perform tasks like summarization, translation, and question-answering.**
- **Fine-tuning and prompting techniques improve model performance.**
- **Training LLMs require significant computational power.**