Name-Prateek Upadhayay

Roll no. - 2K20/AE/053

prateekupadhayay_ae20a15_66@dtu.ac.in

Task 1: GPT-2 Model & Checkpoints (20 Points)

Methodology:

Model Architecture:
- Implemented the GPT-2 model based on the provided GPT-2 paper and the given code.
- Included key components such as token and positional embeddings, transformer layers with multi-head self-attention, and point-wise feed-forward networks.
- Developed a decoder block.

Validation:
- Utilized a random input sequence to perform a forward pass through the model for validation.
- Ensured that the output shape matched expectations.

Resources:
- Referred to the GPT-2 paper and Andrej Karpathy's nanogpt repository for guidance.
- Focused on understanding and implementing the transformer architecture with self-attention mechanisms.

Task 2 | Transformer Architectural Changes (40 Points)

1. **Rotary Positional Embedding:**

- Implementation:
  - Replaced the original positional embeddings with Rotary embeddings as described in the RoFormer paper by Su et al.
  - Incorporated the rotational encoding mechanism into the model's architecture.

2. **Group Query Attention:**

- Implementation:
  - Integrated the Group Query Attention mechanism into the model based on insights from the GQA paper by Ainslie et al.

3. **Sliding Window Attention:**

- Implementation:
    - Incorporated the Sliding Window Attention mechanism into the model following the Longformer paper by Beltagy et al.

Successfully implemented all three requested changes: Rotary Positional Embedding, Group Query Attention, and Sliding Window Attention.

Task 3: Training Loop Implementation (40 Points)

Methodology:

Single GPU Training Loop:
- Defined hyperparameters, loss function, and optimizer.
- Moved the model and data to the GPU if available.
- Implemented a basic training loop with forward pass, backward pass, and optimization.

Distributed Data Parallel (DDP):
- Extended the training loop to support DDP using DistributedDataParallel.
- Ensured proper synchronization and data distribution across multiple GPUs.
- Used torch.distributed.launch for launching DDP.

Fully Sharded Data Parallel (FSDP):
- Utilized the FullyShardedDataParallel module for FSDP implementation.
- Initialized the model and optimizer with FSDP.
- Adapted the training loop to work with FSDP.