Transformer Code Execution Flow

Overview

This document traces the execution flow of the transformer implementation, starting from test.py and following through all the components until the final loss calculation after 50 epochs.

1. Initial Setup in test.py

1. Configuration Loading

• Load TransformerConfig from config.py

• Set model parameters (d_model=512, num_heads=8, etc.)

2. Model Initialization

• Create Transformer instance

• Initialize encoder and decoder stacks

• Set up embeddings and final layer

3. Training Setup

• Create Trainer instance

• Initialize AdamOptimizer

• Set up learning rate scheduler

2. Training Loop Execution

For each epoch (50 epochs total):

1. Forward Pass

a. Transformer.__call__()

• Input embedding and positional encoding

• Pass through encoder stack

• Pass through decoder stack

• Final linear layer projection

2. Encoder Stack Processing

a. For each encoder layer:
• Self-attention computation
• Layer normalization
• Feed-forward network
• Residual connections

3. Decoder Stack Processing
a. For each decoder layer:
• Masked self-attention
• Encoder-decoder attention
• Layer normalization
• Feed-forward network
• Residual connections

4. Loss Computation
• Apply softmax to model output
• Compute cross-entropy loss
• Track loss for visualization

5. Backward Pass
• Compute gradients through all layers
• Update parameters using Adam optimizer
• Update learning rate according to schedule

3. Key Component Interactions

1. Attention Mechanism Flow
• Input → Query/Key/Value projection
• Attention scores computation
• Softmax and dropout
• Output projection

2. Layer Normalization Flow
• Compute mean and variance
• Normalize and scale

• Apply learnable parameters

3. Feed-Forward Network Flow
• First linear transformation
• ReLU activation
• Second linear transformation

4. Residual Connections
• Add input to each sub-layer output
• Helps with gradient flow
• Preserves information

4. Final Results

After 50 epochs:
• Initial loss: ~6.91 (random initialization)
• Final loss: ~5.71
• Learning rate progression:
- Starts small (warmup phase)
- Gradually increases
- Stabilizes for final training

The loss decrease shows that:
• Model is learning effectively
• Architecture is working as intended
• Learning rate schedule is appropriate