Optimizing Attention Mechanisms in Transformers

Week 4: Project Overview & Initial Results

Chandler Cheung, Charis Gao, Jordan Hochman

Background

- Transformer models have become central to NLP tasks
- In recent years, size of models has grown exponentially
- Key challenge: $O(n^2)$ complexity in attention mechanism
- Growing model sizes create memory constraints
- Need more efficient attention mechanism without degrading performance

Optimization Problem

- Develop customizable attention mask that learns important tokens in sequence instead of attending to all tokens
- Train model with optimized attention mask to produce outputs similar to a baseline, unmodified transformer
- Preserve model quality while reducing computational cost

Mathematical Formulation

Core objective: minimize KL-divergence between baseline and custom model

$$\mathcal{L} = \mathrm{KL}\big(P_{\text{base}} \,\|\, P_{\text{custom}}\big)$$

Success Metrics

- Accuracy retention: comparable performance
- Computational improvement: reduced memory and speed gains
- Distribution alignment: low KL-divergence

Current Implementation

- Baseline model: GPT-2 (unoptimized attention mechanism)
- Custom attention module: fixed "last-10-tokens" window
  - Learnable parameters dictating weights for tokens

```python
def forward(self, hidden_states, attention_mask=None, **kwargs):
    ...
    # Create sliding window attention mask
    full_mask = torch.full(
        (batch_size, self.num_heads, seq_length, seq_length),
        float('-inf'),
        device=hidden_states.device
    )
    for i in range(seq_length):
        start_idx = max(0, i - self.window_size)
        full_mask[:, :, i, start_idx:i+1] = 0
    ...
    return self.out_proj(context)
```

Current Implementation

- Loss Computation
  - Compute logits from both models on the same input batch
  - Calculate KL-divergence and minimize

```python
def kl_divergence_loss(logits_custom, logits_ref, mask):
    assert logits_custom.shape == logits_ref.shape, \
        f"Shape mismatch: {logits_custom.shape} vs {logits_ref.shape}"

    log_probs_custom = F.log_softmax(logits_custom, dim=-1)
    probs_ref = F.softmax(logits_ref.detach(), dim=-1)  # Detach reference model

    # Calculate per-token KL
    kl = (probs_ref * (probs_ref.log() - log_probs_custom)).sum(-1)

    # Apply padding mask and average
    active_tokens = mask.sum()
    return (kl * mask).sum() / active_tokens
```

Initial Results - Training Progress

- Over 100 epochs, loss (KL-divergence) decreased from 1.61 to 0.07 on sample data
  - Custom attention can mimic the reference model's distributions
- Tested with a few prompts, resulting in output text mimicing style similar to GPT-2, though often less coherent due to the limited "last-10-tokens" context

Initial Results - Sample Outputs

```
Prompt: Hello, my name is

Reference: ... I am the founder of Inoscular Robotics ...
Custom: ... I have you doing so much easier than ever ...
```

```
Prompt: The meaning of life is

Reference: ... matter's consciousness. True, you can stop ...
Custom: ... a newbies for what, welcome as an earthquake ...
```

- Custom model's outputs sometimes drift or become less coherent
- Roughly follows prompts and produce recognizable English words
- Custom model captures some of GPT-2's output token distribution

Current Limitations

- No dynamic mask optimization
  - Currently using fixed "last-10-tokens" window
- Need to train on larger dataset
  - Currently using small synthetic dataset
- No measure of memory or speed usage

Next Steps

- Implement adaptive mask learning to identify important tokens
- Measure memory usage and speed improvements
- Extend to full WikiText-2 dataset / more training data
- Optimize hyperparameters

Future Ideas

- Look into models besides GPT-2
- Explore alternative approaches: blockwise/local attention, knowledge distillation, etc.

Thank you!

Any questions?