Building a GPT2 Transformer-Based Model From Scratch

1. Introduction

This project implements a simplified GPT-2 model using PyTorch, trained on the TinyStories dataset. The report documents the provided code without external assumptions.

2. Model Implementation

2.1 Core Components (From Code)

- **Implemented Layers:**
  - **PositionalEncoding**: Adds positional information using sine/cosine functions.
  - **MultiHeadAttention**: Multi-head self-attention with causal masking.
  - **FeedForward**: Two-layer MLP with ReLU activation.
  - **TransformerBlock**: Combines attention, feed-forward, LayerNorm, and residual connections.
  - **GPT2**: Integrates all components with token embeddings and a linear output layer.

```python
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, max_len=512):
        super().__init__()
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)  # Shape: [1, max_len, d_model]
        self.register_buffer('pe', pe)

    def forward(self, x):
        return x + self.pe[:, :x.size(1), :]
```

```python
class MultiHeadAttention(nn.Module):
    def __init__(self, d_model, num_heads):
        super().__init__()
        assert d_model % num_heads == 0
        self.d_model = d_model
        self.num_heads = num_heads
        self.d_k = d_model // num_heads
        self.W_q = nn.Linear(d_model, d_model)
        self.W_k = nn.Linear(d_model, d_model)
        self.W_v = nn.Linear(d_model, d_model)
        self.W_o = nn.Linear(d_model, d_model)

        self.scale = math.sqrt(self.d_k)

    def scaled_dot_product_attention(self, Q, K, V, mask=None):
        # Q, K, V shape: [batch_size, num_heads, seq_len, d_k]
        scores = torch.matmul(Q, K.transpose(-2, -1)) / self.scale  # [B, H, T, T]

        if mask is not None:
            # mask shape expected: [batch_size, seq_len, seq_len]
            if mask.dim() == 3:
                mask = mask.unsqueeze(1)  # [B, 1, T, T], to broadcast over heads
            scores = scores.masked_fill(mask == 0, float('-inf'))

        attn = torch.softmax(scores, dim=-1)  # attention weights
        output = torch.matmul(attn, V)         # weighted sum
        return output, attn

    def forward(self, x, mask=None):
        batch_size, seq_len, _ = x.size()

        Q = self.W_q(x)
        K = self.W_k(x)
        V = self.W_v(x)

        # reshape for multi-head
        Q = Q.view(batch_size, seq_len, self.num_heads, self.d_k).transpose(1, 2)
        K = K.view(batch_size, seq_len, self.num_heads, self.d_k).transpose(1, 2)
        V = V.view(batch_size, seq_len, self.num_heads, self.d_k).transpose(1, 2)

        context, attn = self.scaled_dot_product_attention(Q, K, V, mask)
        context = context.transpose(1, 2).contiguous().view(batch_size, seq_len, self.d_model)
        return self.W_o(context), attn
```

```python
# Cell 4: FeedForward
class FeedForward(nn.Module):
    def __init__(self, d_model, d_ff):
        super().__init__()
        self.linear1 = nn.Linear(d_model, d_ff)
        self.linear2 = nn.Linear(d_ff, d_model)
        self.relu = nn.ReLU()

    def forward(self, x):
        return self.linear2(self.relu(self.linear1(x)))
```

```python
# Cell 5: TransformerBlock
class TransformerBlock(nn.Module):
    def __init__(self, d_model, num_heads, d_ff, dropout=0.1):
        super().__init__()
        self.attn = MultiHeadAttention(d_model, num_heads)
        self.norm1 = nn.LayerNorm(d_model)
        self.ff = FeedForward(d_model, d_ff)
        self.norm2 = nn.LayerNorm(d_model)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, mask=None):
        attn_output, attn_weights = self.attn(x, mask)
        x = self.norm1(x + self.dropout(attn_output))
        ff_output = self.ff(x)
        x = self.norm2(x + self.dropout(ff_output))
        return x, attn_weights
```

```python
class GPT2(nn.Module):
    def __init__(self, vocab_size, d_model=128, num_heads=4, num_layers=2, d_ff=512, max_len=512, dropout=0.1):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, d_model)
        self.pos_encoding = PositionalEncoding(d_model, max_len)
        self.transformer_blocks = nn.ModuleList([
            TransformerBlock(d_model, num_heads, d_ff, dropout) for _ in range(num_layers)
        ])
        self.output_layer = nn.Linear(d_model, vocab_size)
        self.dropout = nn.Dropout(dropout)
        self.max_len = max_len

    def forward(self, x, mask=None):
        batch_size, seq_len = x.size()

        x = self.embedding(x)
        x = self.pos_encoding(x)
        x = self.dropout(x)

        attn_weights = []
        if mask is not None:
            # Dataset mask is [batch_size, seq_len], convert to [batch_size, 1, seq_len, seq_len]
            mask = mask.unsqueeze(-1)  # [batch_size, seq_len, 1]
            mask = mask.unsqueeze(1)   # [batch_size, 1, seq_len, 1]
            # Create a causal mask
            causal_mask = torch.triu(torch.ones(seq_len, seq_len, device=x.device), diagonal=1).bool()
            causal_mask = ~causal_mask  # Lower triangle (including diagonal) is True
            # Combine dataset mask with causal mask
            mask = mask * causal_mask.unsqueeze(0)  # [batch_size, 1, seq_len, seq_len]

        for block in self.transformer_blocks:
            x, attn = block(x, mask)
            attn_weights.append(attn)

        logits = self.output_layer(x)
        return logits, attn_weights
```

2.2 Data Processing (From Code)

- **Dataset**: TinyStories (from `/kaggle/input/tinystories/`).
- **Tokenization**: GPT2Tokenizer from Hugging Face.
- **Preprocessing**:
    - Sequences truncated/padded to `max_len=512`.
    - Attention masks used to ignore padding tokens.

```python
# Cell 7: Dataset
class TinyStoriesDataset(Dataset):
    def __init__(self, data_path, max_len=512):
        self.tokenizer = GPT2Tokenizer.from_pretrained("/kaggle/input/gpt2-tokenizer")
        self.tokenizer.pad_token = self.tokenizer.eos_token
        self.max_len = max_len
        with open(data_path, "r", encoding="utf-8") as f:
            self.lines = f.readlines()

    def __len__(self):
        return len(self.lines)

    def __getitem__(self, idx):
        line = self.lines[idx]
        encoding = self.tokenizer(
            line,
            truncation=True,
            max_length=self.max_len,
            padding="max_length",
            return_tensors="pt"
        )
        input_ids = encoding["input_ids"].squeeze()
        attention_mask = encoding["attention_mask"].squeeze()
        return input_ids, attention_mask
```

```python
# Cell 8: Initialize tokenizer and datasets
tokenizer = GPT2Tokenizer.from_pretrained("/kaggle/input/gpt2-tokenizer")
tokenizer.pad_token = tokenizer.eos_token

train_dataset = TinyStoriesDataset("/kaggle/input/tinystories/TinyStories-train-small.txt")
val_dataset = TinyStoriesDataset("/kaggle/input/tinystories/TinyStories-validation-small.txt")

train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=8)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
```

```
Using device: cpu
```

3. Training (From Code)

3.1 Training Setup

- **Model Architecture (From** `GPT2` **Class)**:
  - `d_model=128`, `num_heads=4`, `num_layers=2`, `d_ff=512`.
- **Loss Function**: CrossEntropyLoss (ignores padding tokens).
- **Optimizer**: AdamW (`lr=3e-4`).
- **Hardware**: CPU (as shown in `device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')`).

3.2 Training Process (From `train_model` Function)

- **Epochs**: 5 (as per the code).

- **Batch Size**: 8.
- **Loss Tracking**:
    - `Train Loss` and `Val Loss` computed per epoch using `CrossEntropyLoss`.
    - Results printed in the output during training.

```
# Cell 9: Training Function
def train_model(model, train_loader, val_loader, tokenizer, num_epochs=5, device='cuda'):
    optimizer = torch.optim.Adam(model.parameters(), lr=3e-4)
    loss_fn = torch.nn.CrossEntropyLoss(ignore_index=tokenizer.pad_token_id)

    train_losses = []
    val_losses = []

    model.to(device)

    for epoch in range(num_epochs):
        model.train()
        total_train_loss = 0
        for xb, mask in tqdm(train_loader, desc=f"Epoch {epoch+1} [Training]"):
            xb, mask = xb.to(device), mask.to(device)
            logits, _ = model(xb, mask)

            # Shift input and target for language modeling
            shift_logits = logits[..., :-1, :].contiguous()
            shift_labels = xb[..., 1:].contiguous()

            loss = loss_fn(shift_logits.view(-1, shift_logits.size(-1)),
                           shift_labels.view(-1))

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            total_train_loss += loss.item()

        avg_train_loss = total_train_loss / len(train_loader)
        train_losses.append(avg_train_loss)

        # Validation
        model.eval()
        total_val_loss = 0
        with torch.no_grad():
            for xb, mask in val_loader:
                xb, mask = xb.to(device), mask.to(device)
                logits, _ = model(xb, mask)

                shift_logits = logits[..., :-1, :].contiguous()
                shift_labels = xb[..., 1:].contiguous()

                loss = loss_fn(shift_logits.view(-1, shift_logits.size(-1)),
                               shift_labels.view(-1))
                total_val_loss += loss.item()

        avg_val_loss = total_val_loss / len(val_loader)
        val_losses.append(avg_val_loss)

        print(f"✅ Epoch {epoch+1} | Train Loss: {avg_train_loss:.4f} | Val Loss: {avg_val_loss:.4f}")

    return train_losses, val_losses
```

```
# Cell 10: Initialize and train model
model = GPT2(vocab_size=tokenizer.vocab_size, d_model=128, num_heads=4, num_layers=2, d_ff=512)
model.to(device)
train_losses, val_losses = train_model(model, train_loader, val_loader, tokenizer, num_epochs=5, device=device)
```

```
Epoch 1 [Training]:   1%|          | 1569/277791 [2:51:06<480:02:57,  6.26s/it]
```

4. Evaluation (From Code)

4.1 Quantitative Metrics (From `compute_perplexity` Function)

- **Perplexity**:
    - Calculated on the validation set using:

perplexity = torch.exp(torch.tensor(avg_loss)).item()

- **Note**: No actual results are printed because training was incomplete (the `train_model` cell was still running in the last output).

4.2 Text Generation (From `generate_text` Function)

- **Example in Code**:

```
prompt = "Once upon a time"
generated_text = generate_text(model, tokenizer, prompt, max_len=100, device=device)
```

- **Note**: No generated output is available since training was not completed.

```
# Cell 11: Evaluation and Generation Functions
def compute_perplexity(model, data_loader, device='cuda'):
    model.eval()
    total_loss = 0
    total_tokens = 0
    criterion = nn.CrossEntropyLoss(ignore_index=tokenizer.pad_token_id, reduction='sum')

    with torch.no_grad():
        for input_ids, mask in data_loader:
            input_ids, mask = input_ids.to(device), mask.to(device)
            logits, _ = model(input_ids, mask)

            shift_logits = logits[..., :-1, :].contiguous()
            shift_labels = input_ids[..., 1:].contiguous()

            loss = criterion(shift_logits.view(-1, shift_logits.size(-1)),
                             shift_labels.view(-1))
            total_loss += loss.item()
            total_tokens += (shift_labels != tokenizer.pad_token_id).sum().item()

    avg_loss = total_loss / total_tokens
    perplexity = torch.exp(torch.tensor(avg_loss)).item()
    return perplexity

def top_k_filtering(logits, top_k):
    top_k = min(top_k, logits.size(-1))
    indices_to_remove = logits < torch.topk(logits, top_k)[0][..., -1, None]
    logits[indices_to_remove] = -float('Inf')
    return logits

def generate_text(model, tokenizer, prompt, max_len=100, temperature=0.7, top_k=50, device='cuda'):
    model.eval()
    input_ids = tokenizer.encode(prompt, return_tensors='pt').to(device)
    generated = input_ids

    with torch.no_grad():
        for _ in range(max_len):
            logits, _ = model(generated)
            next_token_logits = logits[:, -1, :] / temperature
            filtered_logits = top_k_filtering(next_token_logits, top_k)
            next_token = torch.multinomial(torch.softmax(filtered_logits, dim=-1), num_samples=1)
            generated = torch.cat((generated, next_token), dim=1)
            if next_token.item() == tokenizer.eos_token_id:
                break

    return tokenizer.decode(generated[0], skip_special_tokens=True)
```

```
# Cell 12: Evaluate and Generate
perplexity = compute_perplexity(model, val_loader, device=device)
print(f"Perplexity: {perplexity:.4f}")

prompt = "Once upon a time"
generated_text = generate_text(model, tokenizer, prompt, max_len=100, device=device)
print("Generated Text:", generated_text)

# Plot training curves
plt.figure(figsize=(8, 6))
plt.plot(range(1, len(train_losses)+1), train_losses, label='Training Loss')
plt.plot(range(1, len(val_losses)+1), val_losses, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

5. Observations

1. **Limitations**:

    a. Small model size (only 2 layers, `d_model=128`) compared to the original GPT-2.
    b. Training on CPU is extremely slow (as seen in the slow progress in the output).
2. **Improvement Points**:
    a. Increase model size (`num_layers`, `d_model`).
    b. Use GPU for faster training.

6. Conclusion

The provided code correctly implements all core GPT-2 components. However, final results are unavailable due to incomplete training. To complete the project:

- **Finish training on GPU.**
- **Record actual** `Train Loss` **and** `Val Loss` **values after each epoch.**
- **Compute final perplexity and text generation examples.**

Team members:-

Bassant Mohammed Saleh 2205178

Naira Waheed Ahmed Ali 2205210

Daie Ali el-chazly 2205211

Noureen Hamdy Mahmoud 2205209