Week 5: Transformers - The Architecture Behind ChatGPT

Pre-Class Discovery & Post-Class Application

How to Use This Handout

- **Before Class**: Complete Part A to build intuition and discover key problems
- **During Class**: Learn the technical solutions to these problems
- **After Class**: Complete Part B to apply and deepen your understanding

---

PART A: PRE-CLASS DISCOVERY

No prior knowledge required - Let's discover why we need transformers!

1 A1: The Speed Problem (10 minutes)

1.1 The Telephone Game

Real World

Imagine passing a message through 100 people in a line, where each person can only whisper to the next person. How long would it take? What could go wrong?

Exercise: Let's think about processing this sentence word by word:

"The student who studied hard and completed all assignments passed the exam"
If you could only read one word at a time, in order:

1. How many steps would it take to read the whole sentence? ____
2. Could you understand "passed" before reading "student"? ____
3. What if you had 12 friends to help - but you still had to read in order? ____

Think: What if you could see ALL words at the same time, like looking at a complete picture?

1.2 Drawing the Difference

Exercise: Draw how you think these two approaches would look:

Approach 1: One word at a time

Approach 2: All words at once

Discovery

You've just discovered the key limitation that transformers solve! Reading sequentially is slow and can lose information. Seeing everything at once is the breakthrough.

2 A2: The Connection Problem (10 minutes)

2.1 Who Talks to Whom?

Intuition

In a classroom, imagine if students could only talk to their immediate neighbors vs. everyone being able to talk to everyone. Which would be better for group understanding?

Exercise: Look at this sentence: "The cat that chased the mouse sat"

Draw lines connecting words that need to "talk" to understand the sentence:

The cat that chased the mouse sat

Q: Which words need to connect to understand who did the sitting?

Q: Which words need to connect to understand what was chased?

2.2 The Spotlight Metaphor

Real World

Imagine each word has a spotlight it can shine on other words. The brightness shows how much attention it pays to that word. "Cat" might shine bright light on "sat" (the action) and dimmer light on "the" (less important).

Exercise: For the word "sat", how bright should its spotlight be on each word? (Use High-/Medium/Low)

"sat" looks at:	**Brightness**
The	_____
cat	_____
that	_____
chased	_____
the	_____
mouse	_____
sat	_____

3 A3: The Multiple Viewpoints Problem (10 minutes)

3.1 Ambiguous Words

Exercise: Consider: "The bank by the river bank collapsed"

The first "bank" could mean:

- A financial institution
- The edge of a river

Q: What different types of information would help you decide?

1. _____
2. _____
3. _____

3.2 Multiple Cameras

Real World

Like filming a scene with multiple cameras: - Camera 1: Shows who's doing what (grammar) - Camera 2: Shows what words mean (meaning) - Camera 3: Shows where words are (position) - Camera 4: Shows the mood (sentiment)

Think: If you had 8 different "cameras" looking at a sentence, what would each look for?

4 A4: The Order Problem (10 minutes)

4.1 When Order Matters

Exercise: These sentences use the exact same words. What's different?

1. "The dog chased the cat" 2. "The cat chased the dog"
If a computer only saw the words without positions, could it tell them apart? ____

Q: How would you tell a computer about word positions? List 3 ideas:

1. _____
2. _____
3. _____

4.2 Position as a Fingerprint

Intuition

What if each position had a unique "fingerprint" - a special pattern that never repeats? Like musical notes at different frequencies creating unique combinations.

Exercise: Design your own position system. How would you mark positions 1, 2, 3?

5 A5: The Stacking Problem (5 minutes)

5.1 Building Understanding

Real World

Like a layer cake where each layer adds more flavor: - Layer 1: Basic word connections - Layer 2: Phrase understanding - Layer 3: Sentence meaning - Layer 4: Paragraph context - And so on...

Q: If one layer gives basic understanding, what happens with 12 layers? 96 layers?

Think: GPT-3 has 96 layers. What might each layer be learning?

Checkpoint

Pre-Class Complete! You've discovered:

- Why sequential processing is limiting
- Why words need to connect directly
- Why multiple viewpoints matter
- Why position information is crucial
- Why stacking layers increases power

Now you're ready to learn HOW transformers solve these problems!

PART B: POST-CLASS APPLICATION

Now that you know the concepts, let's apply them!

6 B1: Attention Mechanism Mathematics (15 minutes)

6.1 Query, Key, Value

Technical Detail

You learned in class:

- Query (Q): What information am I looking for?
- Key (K): What information do I contain?
- Value (V): What actual content do I provide?

Attention scores = $\frac{QK^T}{\sqrt{d_k}}$

Exercise: Complete the attention computation steps:

Given: - Q = Query matrix (what "sat" is looking for) - K = Key matrix (what each word offers) - V = Value matrix (actual information) - $d_k = 64$ (dimension size)

1. Compute similarity: $Q \times K^T$ gives us _____
2. Why divide by $\sqrt{64} = 8$? _____
3. Apply softmax to get _____
4. Multiply by V to get _____

6.2 Calculating Attention Weights

Exercise: Calculate actual attention weights:

If the dot products between "sat" and other words are: - sat · The = 2 - sat · cat = 8 - sat · sat = 6 After scaling by $\sqrt{64} = 8$: - Score(The) = 2/8 = 0.25 - Score(cat) = 8/8 = ___ - Score(sat) = 6/8 = ___ After softmax (approximately): - Weight(The) = 0.20 - Weight(cat) = ___ - Weight(sat) = ___ (Weights must sum to 1.0)

7 B2: Multi-Head Attention Design (10 minutes)

7.1 Head Dimension Calculation

Technical Detail

With multi-head attention: - Model dimension: $d_{model} = 512$ - Number of heads: $h = 8$ - Head dimension: $d_k = d_{model}/h$

Exercise: Calculate dimensions:

Model	Calculations	Head Size
512-dim, 8 heads	512 / 8 =	___
768-dim, 12 heads	_____ =	___
1024-dim, 16 heads	_____ =	___

7.2 Combining Heads

Q: After each head computes attention separately, how do we combine them?

Steps: 1. Each head outputs: _____ 2. Concatenate all heads: _____ 3. Apply linear transformation: _____

8 B3: Positional Encoding (10 minutes)

8.1 Sinusoidal Encoding Formula

Technical Detail

Position encoding uses:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

Exercise: Calculate position encoding for position 1, dimension 0:

Given: pos = 1, i = 0, $d_{model} = 512$

$$PE_{(1,0)} = \sin\left(\frac{1}{10000^{0/512}}\right) \quad (1)$$

$$= \sin\left(\frac{1}{10000^{0}}\right) \quad (2)$$

$$= \sin(\_\_\_) \quad (3)$$

$$= \_\_\_\_\_ \quad (4)$$

Q: Why use different frequencies (the $10000^{2i/d_{model}}$ term)?

9 B4: Complete Transformer Architecture (10 minutes)

9.1 Layer Components

Exercise: Fill in the missing components and their purposes:

Component	Purpose
Multi-Head Attention	_____
Residual Connection	Preserves information, helps gradients
Layer Normalization	_____
Feed-Forward Network	Processes each position independently
Dropout	_____

9.2 Residual Connection Math

Technical Detail

Residual connection: Output = Layer$(x) + x$

Q: If Layer(x) fails (outputs near zero), what happens to the output?

Q: Why is this important for training deep networks?

10 B5: Implementation Exercise (15 minutes)

10.1 Implementing Scaled Dot-Product Attention

Technical Detail

Now implement the attention mechanism you learned in class.

```python
import numpy as np

def softmax(x):
    """Compute softmax values."""
    exp_x = np.exp(x - np.max(x, axis=-1, keepdims=True))
    return exp_x / np.sum(exp_x, axis=-1, keepdims=True)

def scaled_dot_product_attention(Q, K, V):
    """
    Implement scaled dot-product attention.

    Args:
        Q: Query matrix [seq_len, d_k]
        K: Key matrix [seq_len, d_k]
        V: Value matrix [seq_len, d_k]

    Returns:
        output: Attention output [seq_len, d_k]
        weights: Attention weights [seq_len, seq_len]
    """
    d_k = Q.shape[1]

    # Step 1: Compute QK^T
    scores = # YOUR CODE HERE

    # Step 2: Scale by sqrt(d_k)
    scores = # YOUR CODE HERE

    # Step 3: Apply softmax
    weights = # YOUR CODE HERE

    # Step 4: Compute weighted sum of values
    output = # YOUR CODE HERE

    return output, weights

# Test your implementation
seq_len, d_k = 4, 64
Q = np.random.randn(seq_len, d_k)
K = np.random.randn(seq_len, d_k)
V = np.random.randn(seq_len, d_k)

output, weights = scaled_dot_product_attention(Q, K, V)

# Verify
print(f"Output shape: {output.shape}")
print(f"Weights shape: {weights.shape}")
print(f"Weights sum to 1: {np.allclose(weights.sum(axis=1), 1)}")
```

```
print(f"Sample weights:\n{weights[0]}")
```

10.2 Multi-Head Attention

Exercise: Implement multi-head attention splitting:

```
def split_heads(x, num_heads):
    """
    Split last dimension into multiple heads.

    Args:
        x: Input tensor [seq_len, d_model]
        num_heads: Number of attention heads

    Returns:
        Split tensor [num_heads, seq_len, d_k]
    """
    seq_len, d_model = x.shape
    d_k = d_model // num_heads

    # Reshape to [seq_len, num_heads, d_k]
    x = x.reshape(seq_len, num_heads, d_k)

    # Transpose to [num_heads, seq_len, d_k]
    return # YOUR CODE HERE

# Test
x = np.random.randn(10, 512)  # 10 tokens, 512 dimensions
heads = split_heads(x, 8)
print(f"Split shape: {heads.shape}")  # Should be [8, 10, 64]
```

11 B6: Analysis Questions (10 minutes)

11.1 Computational Complexity

Exercise: Compare computational costs:

For a sequence of length n:

- RNN: $O(n)$ sequential steps, can't parallelize
- Transformer self-attention: $O(\_\_\_)$ complexity, fully parallel

Q: What's the trade-off here?

11.2 Long-Range Dependencies

Exercise: Path length analysis:

To connect position 1 to position 100:

- RNN path length: ___ steps
- Transformer path length: ___ step(s)

Q: How does this affect gradient flow during training?

11.3 Memory Requirements

Technical Detail

Attention weights matrix is $n \times n$ for sequence length n.

Exercise: Calculate memory for attention weights:

Sequence Length	Matrix Size	Memory (float32)
100 tokens	100×100	40 KB
512 tokens	___ × ___	___ MB
2048 tokens	___ × ___	___ MB

12 B7: Advanced Concepts (Optional)

12.1 Attention Patterns

Exercise: Identify what type of attention pattern each head might learn:

1. Head attending mostly to previous word: _____
2. Head attending to first token: _____
3. Head attending uniformly: _____
4. Head attending to same word: _____

12.2 Transformer Variants

Q: How would you modify transformers for:

1. Very long sequences (¿10,000 tokens)?
2. Real-time processing?
3. Limited memory devices?

Key Takeaways

Checkpoint

After completing both parts, you understand:
From Pre-Class:

- Why parallel processing beats sequential
- Why direct connections matter
- Why multiple viewpoints are needed
- Why position encoding is crucial

From Post-Class:

- How attention mathematics works (QKV, scaling, softmax)
- How multi-head attention divides and conquers
- How positional encoding uses sinusoids
- How residual connections enable deep networks
- How to implement attention in code

Next Steps

1. Implement a complete transformer block
2. Visualize attention patterns on real text
3. Explore pre-trained models (BERT, GPT)
4. Read the original "Attention Is All You Need" paper

Real World

You now understand the architecture powering ChatGPT, Claude, and modern AI. From discovering the problems to implementing solutions - you've mastered transformers!