Week 5: Attention Is All You Need - Transformers

Discovery-Based Learning with Guided Exploration

Learning Journey

Today we'll discover the architecture behind ChatGPT, BERT, and modern AI. Imagine a classroom where everyone can talk to everyone else simultaneously - that's the transformer revolution!

Learning Objectives

By the end of this session, you will:

- Understand why RNNs have fundamental limitations
- Discover self-attention from first principles
- Design the multi-head attention mechanism
- Build a complete transformer architecture
- Master positional encodings

---

1 Part 1: The Sequential Processing Problem (10 minutes)

1.1 The Telephone Game

Theory Hint

RNNs process sequences one element at a time, like a game of telephone. Each step depends on the previous one, creating a bottleneck. Information can degrade as it passes through many steps, and we can't process step 10 until we've processed steps 1-9.

Real World Example

RNN (Sequential): Like reading a book one word at a time with your finger
Transformer (Parallel): Like seeing the whole page at once

Exercise: Let's process this sentence with an RNN. Mark the dependencies:

"The student who studied hard and completed all assignments passed the exam"

Word	Step	Depends on previous steps?
The	1	No
student	2	Yes (step 1)
who	3	_____
studied	4	_____
hard	5	_____
and	6	_____
completed	7	_____
all	8	_____
assignments	9	_____
passed	10	_____
the	11	_____
exam	12	_____

Q: Can we process "passed" before we process "assignments"? Why or why not in an RNN?

1.2 Parallelization Challenge

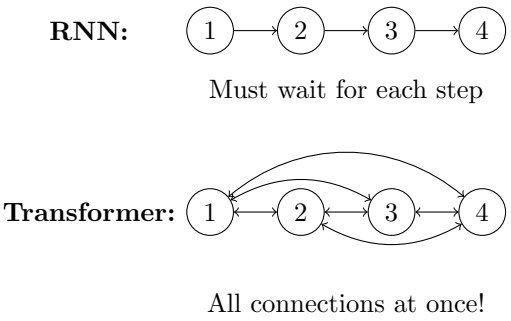
Theory Hint

Modern GPUs excel at parallel computation. If we can process all positions simultaneously instead of sequentially, we can achieve massive speedups. Think about how many operations you could do at once if they didn't depend on each other!

Think: You have 12 GPUs. How many can you use simultaneously to process this sentence with an RNN?

Answer: _____

Q: What if you could look at ALL words at once instead of sequentially?



Discovery

You've identified the key limitation of RNNs: sequential processing prevents parallelization. Transformers process all positions simultaneously!

2 Part 2: Inventing Self-Attention (15 minutes)

2.1 The Direct Connection Idea

Theory Hint

Instead of information flowing step-by-step through hidden states, what if every word could directly "attend" to every other word? This is like giving each word the ability to look at all other words and decide which ones are most relevant to understanding its meaning.

Real World Example

Think of attention like a **spotlight in a theater**:

- Each word is an actor on stage
- Each actor has a spotlight they control
- They can shine their spotlight on other actors (or themselves)
- The brightness shows how much they're paying attention

Exercise: For the sentence "The cat sat", draw arrows showing which words should connect:

The cat sat

Q: How many connections did you draw? For a sentence with n words, how many connections would we need?

Connections = _____

2.2 Computing Relevance

Theory Hint

To decide how much one word should "attend" to another, we compute a relevance score. These scores are then normalized (using softmax) so they sum to 1.0, creating a probability distribution over the input sequence. This tells us what percentage of attention to give each word.

Exercise: For each word pair, assign a relevance score (0-1):

When processing "sat", how relevant is each word?

Query: "sat"	Key	Relevance Score
sat looks at →	The	___
sat looks at →	cat	___
sat looks at →	sat	___
Total		Should sum to 1.0

Think: How would you compute these scores mathematically?

2.3 The Three Roles: Query, Key, Value

Theory Hint

Each word needs to play three distinct roles in attention:

- **Query**: What information am I looking for?
- **Key**: What information do I contain?
- **Value**: What actual content do I provide?

This is similar to a database or dictionary lookup system.

Real World Example

Imagine a library:

- **Query**: "I'm looking for books about cats"
- **Key**: Each book's catalog card (title, subject)
- **Value**: The actual book content

You compare your query to all keys, then take the values of matching books!

Q: Complete the three roles each word plays:

1. **Q**_____: The word asking "who is relevant to me?"
2. **K**___: The word answering "here's what I offer"
3. **V**____: The word providing "here's my actual information"

2.4 The Attention Formula

Theory Hint

The attention mechanism has four steps: 1. Compute similarity between Query and Keys (dot product) 2. Scale to prevent gradient problems (divide by $\sqrt{d_k}$) 3. Normalize with softmax to get weights summing to 1 4. Weight the Values by these attention weights

Exercise: Design the attention mechanism. Fill in the steps:

1. Compute similarity: $Q \times K^T =$ _____
2. Scale by: $\frac{1}{\sqrt{\text{___}}}$ (why scale?)
3. Apply _____ to get weights that sum to 1
4. Multiply by _____ to get final output

Discovery

Congratulations! You just invented self-attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

3 Part 3: Why Multiple Heads? (10 minutes)

3.1 Different Types of Relationships

Theory Hint

A single attention pattern might focus on syntax OR semantics OR position, but rarely all three effectively. By having multiple "heads" of attention running in parallel, each can specialize in detecting different types of relationships between words.

Real World Example

Like having multiple cameras filming a scene:

- Camera 1: Focuses on the main actor (subject)
- Camera 2: Captures the action (verb)
- Camera 3: Shows the setting (context)
- Camera 4: Tracks emotions (sentiment)

Each "head" captures different relationships!

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

First "bank" should attend to different words for different reasons:

- Syntactic: "bank" is a _____ (noun/verb)
- Semantic: "bank" means _____ (financial/shore)
- Position: "bank" is the _____ word

Q: Can a single attention pattern capture all these relationships?

3.2 Multi-Head Design

Theory Hint

Multi-head attention splits the model's representation into multiple subspaces. If we have 8 heads and a model dimension of 512, each head works with 64 dimensions. After each head computes its attention, we concatenate all outputs and mix them with a linear transformation.

Think: What if we had multiple attention mechanisms running in parallel, each looking for different patterns?

Exercise: Design multi-head attention:

1. Number of parallel attentions: _____ (typically 8-16)
2. Each head size: $\frac{d_{model}}{\text{_____}}$
3. How to combine outputs: _____

Visual example of 4 heads looking at "bank":

Head	Focus	Attends to
Head 1	Syntax	"The" (determiner)
Head 2	Meaning	"river" (context)
Head 3	Position	nearby words
Head 4	Topic	"water", "flow"

Discovery

Multi-head attention lets the model attend to different types of information simultaneously - syntax, semantics, position, and more!

4 Part 4: The Position Problem (10 minutes)

4.1 Order Blindness

Theory Hint

Self-attention treats a sentence as a "bag of words" - it doesn't inherently know word order. Without position information, "cat chased mouse" and "mouse chased cat" would look identical to the model. We need to inject position information somehow.

Exercise: Self-attention treats these as identical. Why is this a problem?

1. "The cat chased the mouse"
2. "The mouse chased the cat"

Both have the same words, same attention scores between words...

Q: What information is self-attention missing?

Write two sentences using the same words but different order, where meaning completely changes:
1. _____
2. _____

4.2 Encoding Position

Theory Hint

We need to add position information to each word embedding. The challenge: how to encode positions so that:

- Each position is unique
- The model can understand relative positions (word 3 is closer to word 4 than to word 10)
- It generalizes to sequences longer than training examples

Think: How can we tell the model about word positions?

Exercise: Evaluate these approaches:

Approach	Pros	Cons
Add position number [1,2,3...]	Simple	_____
Learn position embeddings	Flexible	_____
Use sin/cos waves	_____	Complex

4.3 Sinusoidal Encoding

Theory Hint

Sinusoidal (sine/cosine) encodings are clever: they create unique patterns for each position using waves of different frequencies. Lower frequencies change slowly (capturing global position), while higher frequencies change quickly (capturing local distinctions). This allows the model to learn both absolute and relative positions.

Real World Example

Like giving each word a unique "address":

- Position 1: [0.84, 0.54, 0.00, 1.00, ...]
- Position 2: [0.91, 0.42, 0.84, 0.54, ...]
- Position 3: [0.14, -0.99, 0.91, 0.42, ...]

Each position has a unique pattern, like a fingerprint!

Q: Why use sine and cosine for positions?

Properties to achieve:

- Each position should be unique
- Model should understand relative positions
- Should work for any sequence length

Exercise: Fill in the position encoding formula:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\text{-----}}}\right)$$

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

Discovery

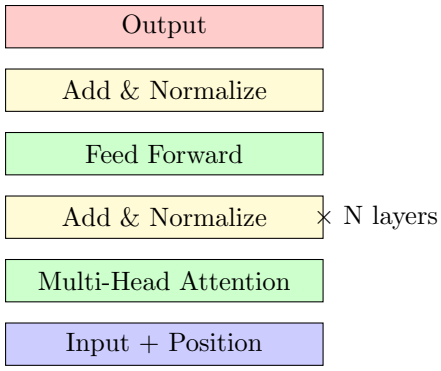
Sinusoidal position encodings allow the model to learn relative positions and generalize to longer sequences than seen during training!

5 Part 5: Building the Complete Transformer (15 minutes)

5.1 The Layer Cake Architecture

Theory Hint

A transformer is built from stacking identical layers. Each layer has two main sub-layers: 1. Multi-head self-attention (for looking at other positions) 2. Position-wise feed-forward network (for processing each position independently) Both use residual connections and layer normalization for stable training.



Exercise: Design one transformer layer. What components do we need?

1. _____ (computes attention)
2. _____ (adds shortcut)
3. _____ (normalizes)
4. _____ (processes each position)
5. _____ (another shortcut)
6. _____ (normalizes again)

5.2 The Feed-Forward Network

Theory Hint

After attention aggregates information across positions, we need to process each position individually. The FFN is a simple 2-layer neural network applied to each position separately. It typically expands the dimension (e.g., 512 → 2048) then contracts it back, allowing complex transformations.

Q: After attention, why do we need position-wise feed-forward networks?

Think about:

- Attention combines information from different positions
- FFN processes _____

5.3 Residual Connections

Theory Hint

Residual connections (also called skip connections) add the input directly to the output: output = layer(x) + x. This creates "highways" for gradients to flow backward and information to flow forward, making it possible to train very deep networks.

Real World Example

Like having both stairs AND an elevator in a building:

- Stairs = Going through the transformation
- Elevator = Direct connection (residual)

Information can take either path!

Exercise: Why add the input to the output (residual/skip connections)?

Benefits:

1. Gradient flow: _____
2. Information preservation: _____
3. Training stability: _____

5.4 Stack and Scale

Q: If one layer is good, what about many layers?

Model	Layers	Parameters
BERT-Base	12	110M
GPT-2	___	1.5B
GPT-3	___	175B

Checkpoint

Like a layer cake, each layer adds more understanding. GPT-3 has 96 layers! Transformers scale exceptionally well - more layers and parameters consistently improve performance.

6 Part 6: Advantages Analysis (10 minutes)

6.1 Parallelization

Theory Hint

Since all positions can be processed simultaneously, transformers can fully utilize modern parallel hardware (GPUs/TPUs). This is a massive advantage over sequential models.

Exercise: Compare processing time:

100-word sequence:

- RNN: 100 sequential steps = ____ time units
- Transformer: ___ parallel step(s) = ____ time unit(s)

Speedup factor: ____×

6.2 Long-Range Dependencies

Theory Hint

In RNNs, information must pass through all intermediate steps. In transformers, any word can directly attend to any other word in just one step, making it much easier to capture long-range dependencies.

Q: How many steps for word 1 to influence word 100?

- RNN: ____ steps (through all intermediate)
- Transformer: ____ step(s) (direct connection)

6.3 Interpretability

Think: With attention weights, what can we visualize?

7 Coding Challenge: Build Your Own Attention

Theory Hint

Let's implement the attention mechanism step by step. Remember:

- Scores = $Q \times K^T$ (dot product for similarity)
- Scale by $\sqrt{d_k}$ to prevent large values
- Softmax normalizes scores to sum to 1
- Final output weights Values by attention scores

```python
import numpy as np

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

def self_attention(Q, K, V):
    """
    Compute scaled dot-product attention.
    Q, K, V: matrices of shape (seq_len, d_k)
    Returns: output matrix and attention weights
    """
    d_k = Q.shape[1]

    # Step 1: Compute scores (Q x K^T)
    scores = # YOUR CODE: np.dot(Q, K.T)

    # Step 2: Scale by sqrt(d_k)
    scores = scores / # YOUR CODE: np.sqrt(d_k)

    # Step 3: Apply softmax to get attention weights
    weights = # YOUR CODE: softmax(scores)

```

```
    # Step 4: Apply weights to values
    output = # YOUR CODE: np.dot(weights, V)

    return output, weights

# Test your implementation!
seq_len, d_k = 4, 8
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 = self_attention(Q, K, V)
print("Attention weights shape:", weights.shape)
print("Output shape:", output.shape)
print("Do weights sum to 1?", np.allclose(weights.sum(axis=1), 1))

# Visualize attention weights
print("\nAttention weights (rounded):")
print(np.round(weights, 2))
```

8 Practice Problems

8.1 Problem 1: Attention Weights

Theory Hint

When filling in attention weights, think about which words are most relevant to each other. Subject words often attend to their verbs, adjectives attend to their nouns, and pronouns attend to what they refer to. Remember: each row must sum to 1.0!

Given the sentence "Dogs love treats", fill in reasonable attention weights:

From ↓ To →	Dogs	love	treats
Dogs	0.5	___	___
love	___	0.2	___
treats	___	___	0.4

8.2 Problem 2: Parallelization

Theory Hint

RNNs must process sequentially: step 1, then step 2, then step 3, etc. Transformers can process all positions at once. This parallelization is what makes transformers so fast on modern hardware.

Calculate the speedup:

- Sentence length: 50 words
- RNN: Processes 1 word per time step
- Transformer: Processes all words at once

Time for RNN: ____ steps Time for Transformer: ____ step(s) Speedup: ____ times faster

8.3 Problem 3: Design Challenge

Design a 2-head attention system for understanding "Time flies like an arrow":

- Head 1 focuses on: _____
- Head 2 focuses on: _____

9 Reflection Questions

Q: Why is the paper titled "Attention Is All You Need"?

Q: What tasks beyond NLP could benefit from transformers?

Q: What are potential limitations of transformers?

Think about: computational cost, memory requirements, sequence length limits...

Key Takeaways

Checkpoint

Remember these essential points:

1. Transformers process all words **in parallel** (unlike sequential RNNs)
2. Self-attention creates **direct connections** between all word pairs
3. Multiple heads capture **different types of relationships**
4. Position encoding tells the model about **word order**
5. Residual connections and normalization enable **deep networks**
6. This architecture powers **ChatGPT, BERT, and modern AI**!

Fun Facts

- The transformer paper has been cited over 100,000 times!
- GPT-3 would take 355 years to train on a single GPU
- Transformers work with images (ViT), music, and even DNA sequences
- The original transformer was trained for translation in just 3.5 days
- The name comes from "transforming" one sequence to another

Next Steps

1. Try the Jupyter notebook to build your own transformer
2. Experiment with attention visualizations
3. Explore pre-trained models next week
4. Challenge: Read "Attention Is All You Need" paper

Real World Example

You now understand the architecture that revolutionized AI. These concepts you discovered today power ChatGPT, BERT, and virtually all modern NLP systems. You're ready to dive deeper into the world of transformers!

Next Week: We'll explore pre-trained language models (BERT, GPT) and the revolution they started!