Natural Language Processing Course

Week 7: Advanced Transformers

Joerg R. Osterrieder
www.joergosterrieder.com

Week 7

Advanced Transformers

The Race to 1 Trillion Parameters

The Accidental Discovery That Changed Everything

OpenAI's experiment (2020):[1]

"What happens if we just... make BERT 100x bigger?"

Expected:

- Slightly better performance
- Diminishing returns
- Waste of compute

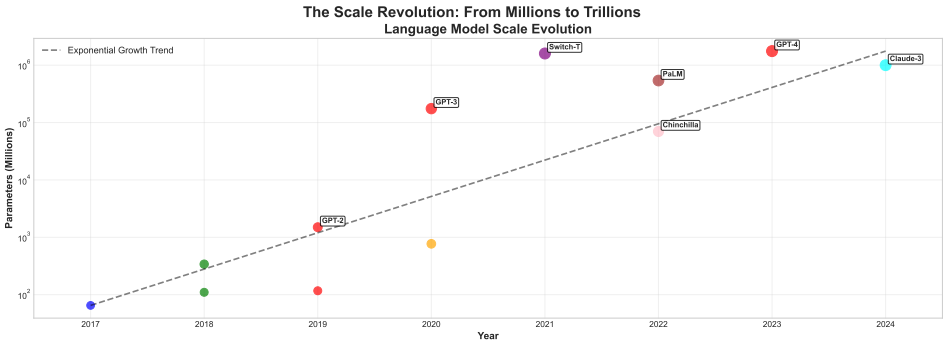
What actually happened:

- Model started doing tasks it was NEVER trained for
- Could translate languages without translation training
- Solved math problems without math training
- Wrote code without code training!

Emergent abilities: Skills that appear only at massive scale

[1] Brown et al. (2020). "Language Models are Few-Shot Learners" (GPT-3)

The Mind-Boggling Scale of Modern LLMs



Advanced Transformers in Production (2024)

Consumer Products:

- ChatGPT: 180M+ users[3]
- GitHub Copilot: 1.2M+ subscribers
- Claude: Advanced reasoning
- Bard/Gemini: Multimodal

Enterprise Applications:

- Code generation: 40% productivity boost[4]
- Document analysis
- Customer service automation
- Content creation at scale

Key Capabilities:

- Zero-shot task solving
- In-context learning
- Chain-of-thought reasoning
- Multimodal understanding

Scale Advantages:

- Better factual knowledge
- Stronger reasoning
- More creative outputs
- Fewer hallucinations[5]

2024: Every major tech company has a 100B+ parameter model

[1] OpenAI statistics, 2024
[2] GitHub Copilot productivity study
[3] Though hallucination remains a challenge

Week 7: What You'll Master

By the end of this week, you will:

- **Understand** why scale unlocks new capabilities
- **Master** architectural improvements for scale
- **Implement** efficient attention mechanisms
- **Explore** few-shot and zero-shot learning
- **Apply** modern prompting techniques

Core Insight: Scale isn't just "bigger" - it's qualitatively different

Emergent Abilities: Magic or Science?

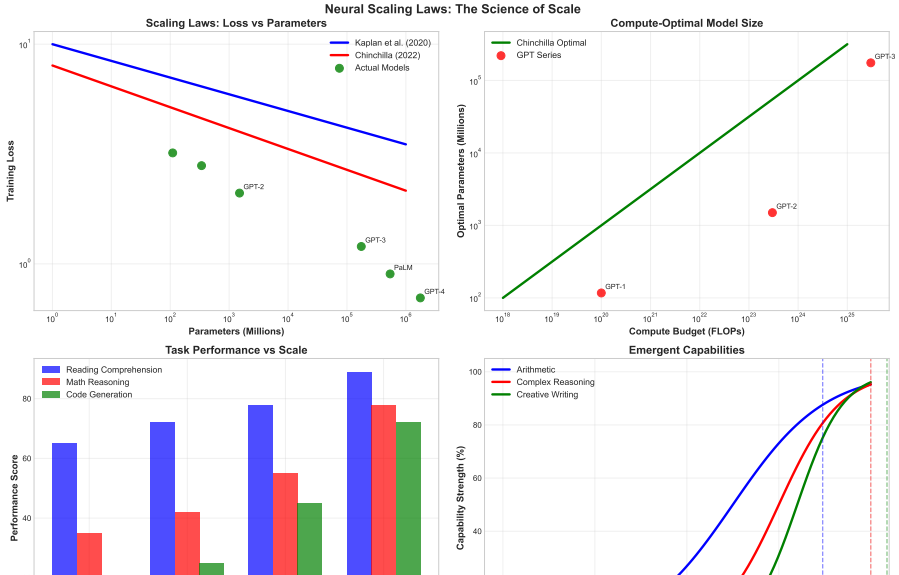
Small models can't, large models can:[6]

1. **Three-digit arithmetic**
   - <10B parameters: 0% accuracy
   - 100B parameters: 90% accuracy
   - No arithmetic in training objective!
2. **Chain-of-thought reasoning**
   - "Let's think step by step..."
   - Small models: Output nonsense
   - Large models: Logical reasoning emerges
3. **In-context learning**
   - Show 3 examples, model learns pattern
   - No gradient updates!
   - Pattern recognition at massive scale

Phase transitions: Abilities appear suddenly at specific scales

[6] Wei et al. (2022). "Emergent Abilities of Large Language Models"

The Scaling Laws: Predictable Progress



Architectural Innovations for Scale

Problem: Attention is $O(n^2)$ - breaks at long sequences

Solutions developed:

1. **Sparse Attention (GPT-3):**
   - Only attend to local + global tokens
   - Reduces to $O(n\sqrt{n})$
   - Maintains performance
2. **Flash Attention (2022):**[8]
   - IO-aware algorithm
   - 2-4x faster, less memory
   - Enables 100k+ context
3. **Rotary Position Embeddings (RoPE):**
   - Better length extrapolation
   - Used in LLaMA, GPT-Neo
   - Relative positions naturally

Innovation focus: Make attention practical at massive scale

[8] Dao et al. (2022). "FlashAttention: Fast and Memory-Efficient Exact Attention"

Implementing Sparse Attention

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class SparseAttention(nn.Module):
    def __init__(self, n_heads, seq_len, sparsity_factor=8):
        # Sparse attention for long sequences
        super().__init__()
        self.n_heads = n_heads
        self.seq_len = seq_len
        self.sparsity = sparsity_factor

        # Define sparse pattern
        self.register_buffer('mask', self.create_sparse_mask(seq_len
            ))

    def create_sparse_mask(self, seq_len):
        # Create sparse attention pattern
        mask = torch.zeros(seq_len, seq_len)

        for i in range(seq_len):
            # Attend to self
            mask[i, i] = 1

            # Local attention (previous k tokens)
            for j in range(max(0, i - self.sparsity), i):
                mask[i, j] = 1

            # Strided attention (every k-th token)
            for j in range(0, i, self.sparsity):
                mask[i, j] = 1

            # Global attention (first few tokens)
            for j in range(min(self.sparsity, seq_len)):
                mask[i, j] = 1
```

Sparse Patterns:

- Local: Nearby tokens (capture local context)
- Strided: Every k-th token (long-range)
- Global: First tokens (task instructions)

Benefits:

- Memory: $O(n)$ instead of $O(n^2)$
- Speed: 10x faster on long sequences
- Quality: 95% of dense performance

Used in:

- GPT-3: Custom sparse patterns
- BigBird: Random + sliding window
- Longformer: Task-specific sparsity

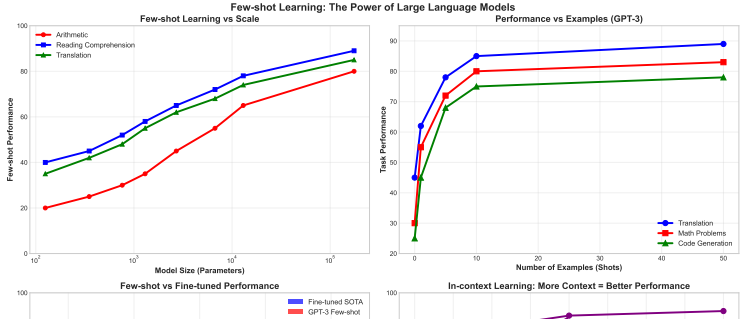
Few-Shot Learning: Teaching by Example

The GPT-3 breakthrough: Learning from context alone

Example - Sentiment Analysis:
```
Review: "The movie was fantastic!" Sentiment: Positive
Review: "Terrible waste of time." Sentiment: Negative
Review: "Best film I've seen all year!" Sentiment: Positive
Review: "Boring and predictable." Sentiment: [Model predicts: Negative]
```

No fine-tuning needed! The model:

- Recognizes the pattern from examples
- Applies it to new inputs
- Works for ANY task with clear examples



Prompt Engineering: The New Programming

Discovery: HOW you ask matters as much as WHAT you ask

Example - Math Problem:

Bad prompt: "What is 37 * 48?"

Result: Often wrong

Good prompt: "Let's solve 37 * 48 step by step: First, break it down..."

Result: Much more accurate!

Effective techniques:

1. **Chain-of-thought:** "Let's think step by step"
2. **Role prompting:** "You are an expert mathematician"
3. **Format specification:** "Answer in JSON format"
4. **Self-consistency:** Generate multiple answers, vote

Prompt engineering: 10x performance difference on same model!

Training at Scale: The Engineering Challenge

Training GPT-3 scale models:

The numbers:

- Parameters: 175 billion
- Training data: 45TB of text
- Compute: 3.14×10^{23} FLOPs
- Cost: $4.6 million[9]
- Time: 34 days on 10,000 GPUs

Technical challenges:

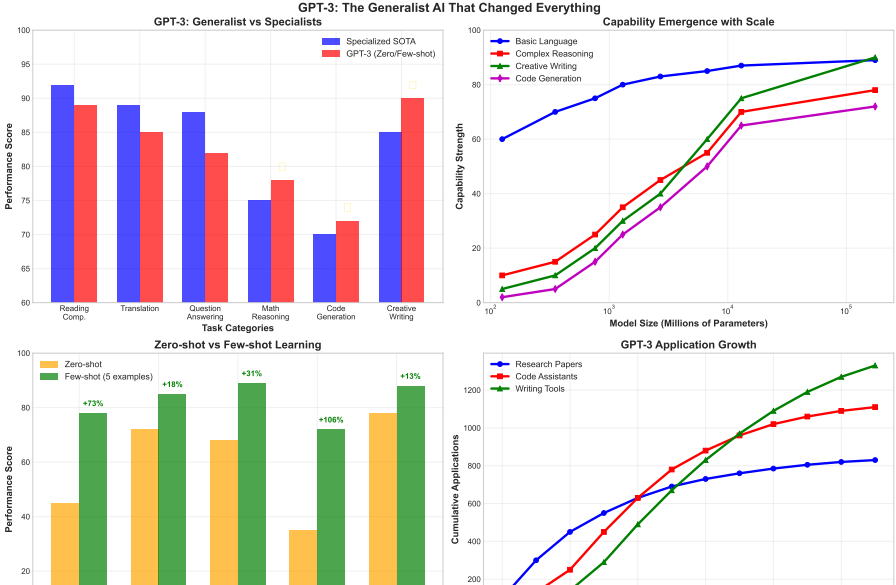
1. **Model parallelism:** Split layers across GPUs
2. **Pipeline parallelism:** Micro-batches through layers
3. **Data parallelism:** Different batches per GPU
4. **Mixed precision:** FP16 with FP32 master weights
5. **Gradient checkpointing:** Trade compute for memory

One bug = $100,000 wasted. Debugging at scale is critical!

[9] Lambda Labs estimate based on cloud GPU pricing

Model Parallelism: Splitting Across GPUs

What Scale Enables: GPT-3 Capabilities



The Large Model Landscape (2024)

Closed Models:

- GPT-4: Multimodal, 1.76T[10]
- Claude 3: Constitutional AI
- Gemini Ultra: 1.75T parameters
- ChatGPT: Continually updated

Open Models:

- LLaMA 2: 7B-70B, efficient
- Mistral: 7B beats 30B models
- Falcon: 180B, permissive license
- BLOOM: 176B, multilingual

Specialized Giants:

- Codex: Programming focus
- Med-PaLM: Medical expertise
- Galactica: Scientific knowledge
- Bloomberg GPT: Financial

Efficiency Focus:

- Chinchilla: Better data scaling
- LLaMA: Quality at smaller size
- Alpaca: Efficient fine-tuning
- QLoRA: 4-bit quantization

2024 trend: "Small" 7B models matching 2020's 175B performance

[10] Parameter counts estimated from capabilities

Pushing the Boundaries: What's Next?

Current frontiers:

1. **Multimodal models:**
   - GPT-4V: Understands images
   - Gemini: Audio, video, images, text
   - DALL-E 3: Integrated generation
2. **Longer context:**
   - Claude: 100k tokens (75k words)
   - GPT-4 Turbo: 128k tokens
   - Research: 1M+ tokens (entire books)
3. **Reasoning improvements:**
   - Chain-of-thought built-in
   - Tool use and function calling
   - Self-correction mechanisms
4. **Efficiency at scale:**
   - Mixture of Experts (MoE)
   - Sparse models: 1T params, 100B active
   - Better architectures than transformers?

Week 7 Exercise: Explore Emergent Abilities

Your Mission: Discover what large models can do that small ones can't

Part 1: Scale Comparison

- Use models of different sizes (GPT-2 vs GPT-3.5)
- Test arithmetic, reasoning, translation
- Document where abilities emerge
- Plot performance vs model size

Part 2: Few-Shot Learning

- Create custom tasks with 0, 1, 3, 5 examples
- Test on: classification, generation, reasoning
- Measure how examples improve performance
- Find optimal number of shots

Part 3: Prompt Engineering

- Compare basic vs chain-of-thought prompts
- Test role prompting effectiveness
- Try self-consistency voting
- Quantify prompt impact

You'll discover: Why scale isn't just "bigger" - it's different!

Key Takeaways: The Scale Revolution

What we learned:

- Scale enables emergent abilities
- Few-shot learning works at large scale
- Architectural innovations enable efficiency
- Prompt engineering is crucial
- Engineering challenges are immense

The paradigm shifts:

Fine-tuning everything $\rightarrow$ Few-shot learning
Model architecture $\rightarrow$ Scale + data
Programming $\rightarrow$ Prompt engineering

Why it matters:

- Democratizes AI capabilities
- Enables new applications
- Changes how we interact with computers

Next week: Tokenization and Subword Models
How do these models handle any text in any language?

References and Further Reading

Foundational Papers:

- Brown et al. (2020). "Language Models are Few-Shot Learners" (GPT-3)
- Kaplan et al. (2020). "Scaling Laws for Neural Language Models"
- Wei et al. (2022). "Emergent Abilities of Large Language Models"

Technical Advances:

- Dao et al. (2022). "FlashAttention"
- Hoffmann et al. (2022). "Training Compute-Optimal LLMs" (Chinchilla)
- Touvron et al. (2023). "LLaMA: Open and Efficient Foundation Models"

Practical Resources:

- "GPT-3 Powers and Limits" - OpenAI blog
- "The Illustrated GPT-3" - Jay Alammar
- Anthropic's research on scaling