Advanced Transformers

Week 7 - The Predictable Path to Intelligence

NLP Course 2025

October 27, 2025

Two-Tier BSc Discovery Presentation

The Scaling Laws Paradox

Three Paths to Better Transformers

Path 1: Bigger

GPT-3 (2020)

- 175B parameters
- 1000× GPT-1
- Emergent abilities
- $4.6M training cost

Bet: Scale alone works

Path 2: Smarter

Mixture of Experts

- 1.6T parameters
- Only 10B active
- Sparse activation
- Same cost as 100B

Bet: Efficiency matters

Path 3: Efficient

Reformer, Linformer

- Reduce $O(n^2)$
- Long sequences
- Lower memory
- Practical deployment

Bet: Algorithms matter

All three work! Next 17 slides explore each path

Different paths for different constraints - no single winner

The Kaplan Scaling Laws (2020)

Scaling Laws: The Mathematics

The Three Power Laws:

$$L(N) = (N_c/N)^{\alpha_N}$$
$$L(D) = (D_c/D)^{\alpha_D}$$
$$L(C) = (C_c/C)^{\alpha_C}$$

where:

- N: Number of parameters
- D: Dataset size (tokens)
- C: Compute (FLOPs)
- $\alpha_N \approx 0.076$
- $\alpha_D \approx 0.095$
- $\alpha_C \approx 0.050$

Key Discovery:
Smooth, predictable improvement

Worked Example:
GPT-3: $N = 175$B, $L = 2.1$ nats

Predict GPT-4 ($N = 500$B hypothetical):

$$\begin{aligned} L(500B) &= L(175B) \times (175B/500B)^{0.076} \\ &= 2.1 \times (0.35)^{0.076} \\ &= 2.1 \times 0.92 \\ &= 1.93 \text{ nats} \end{aligned}$$

Interpretation:
8% loss reduction from 3x parameters
Matches observed scaling!

These laws held from 1M to 175B parameters - remarkable consistency

Emergent Abilities at Scale

Chinchilla: Most Models Are Undertrained

The Discovery (2022):
GPT-3: 175B params, 300B tokens

Chinchilla: 70B params, 1.4T tokens

Result: Chinchilla beats GPT-3!

The Rule:

$$\text{Optimal tokens} \approx 20 \times \text{parameters}$$

GPT-3 should have seen:
$175B \times 20 = 3.5T$ tokens
Actually saw: 300B (9% of optimal!)

Why This Matters:

- Most large models undertrained
- Training longer is cheaper than bigger model
- Compute allocation matters

Practical Impact:

- LLaMA-2 (70B): Trained on 2T tokens
- Follows Chinchilla scaling
- Outperforms GPT-3 with 2.5x fewer params
- Cheaper to run, same quality

Lesson:
Don't just make models bigger - train them longer!

Chinchilla changed how we think about model training

Worked Example: Compute-Optimal Model

Given: Fixed compute budget $C = 10^{23}$ FLOPs

Question: How many parameters N and tokens D to use?

Chinchilla Formula:
For optimal allocation:

$$N_{opt} \approx 0.73 \times C^{0.37}$$
$$D_{opt} \approx 1.45 \times C^{0.37}$$

Calculate:

$$N_{opt} = 0.73 \times (10^{23})^{0.37} = 0.73 \times 1.86 \times 10^{8}$$
$$\approx 136M \text{ parameters}$$

$$D_{opt} = 1.45 \times (10^{23})^{0.37} \approx 270M \text{ tokens}$$

Verification: $D_{opt}/N_{opt} = 270M/136M \approx 2$ (roughly $20\times$ rule)

Optimal compute allocation: Balance parameters and training data

GPT-3: The Scale Breakthrough

GPT-3 Architecture Details

Specifications:

- **Layers**: 96 decoder layers
- **Hidden size**: 12,288 dimensions
- **Attention heads**: 96 (128 dim each)
- **Context window**: 2048 tokens
- **Parameters**: 175 billion
- **Vocabulary**: 50,257 (BPE)

Why So Big:
Scale enables emergent abilities

Training:

- **Dataset**: Common Crawl (570GB)
- **Tokens**: 300 billion
- **Batch size**: 3.2M tokens
- **Hardware**: 10,000+ GPUs
- **Time**: Several months
- **Cost**: $4.6 million

Notable:
8 different model sizes tested (125M to 175B)
All follow same scaling law!

Largest dense transformer ever trained (as of 2020)

Few-Shot Learning: Emergent at Scale

The Cost of Scale

Training Cost Escalation:

Model	Cost
GPT-1 (117M)	\$50K
GPT-2 (1.5B)	\$500K
GPT-3 (175B)	\$4.6M
GPT-4 (est 1.7T)	\$50M+

100× parameters ≈ 1000× cost

Diminishing Returns:
Each 10× scale:

- Cost: 10× increase
- Loss: 0.076 decrease (8%)
- Linear cost, log gains

Why Continue Scaling:

- Emergent abilities worth the cost
- Few-shot learning transforms UX
- Reasoning capabilities
- Multimodal integration (GPT-4)

Alternative Strategies:

- Chinchilla: Train longer, not bigger
- MoE: Sparse activation
- Efficient: Reduce $O(n^2)$
- Distillation: Compress after training

The Tension:
Better models vs affordable training

Scaling works but is expensive - motivates smarter approaches

Worked Example: Predicting Future Performance

Given: GPT-3 at 175B params has loss $L = 2.1$ nats

Question: What loss would 500B parameter model achieve?

Using Scaling Law: $L(N) = L_0 \times (N_0/N)^{\alpha}$ where $\alpha = 0.076$

Step 1: Set up equation

$$L(500B) = 2.1 \times (175B/500B)^{0.076}$$

Step 2: Calculate ratio

$$\frac{175}{500} = 0.35$$

Step 3: Apply exponent

$$0.35^{0.076} \approx 0.92$$

Step 4: Final loss

$$L(500B) = 2.1 \times 0.92 = 1.93 \text{ nats}$$

Interpretation: $3\times$ parameters $\rightarrow$ 8% loss reduction (diminishing returns!)

Mixture of Experts: Sparse Activation

Mixture of Experts: How It Works

Components:

- **Router**: Gating network (small NN)
- **Experts**: E specialist FFNs
- **Top-k selection**: Use best k experts

Forward Pass:

1. Router computes scores for all experts
2. Select top-k (typically $k = 2$)
3. Activate only selected experts
4. Weighted sum of expert outputs

Example:

- 128 experts total
- Top-2 selection
- Activate: $2/128 = 1.6\%$

Gating Function:

$$G(x) = \text{softmax}(\text{KeepTopK}(W_g \cdot x, k))$$

Output:

$$y = \sum_{i \in \text{Top-k}} G(x)_i \cdot E_i(x)$$

Load Balancing:
Auxiliary loss ensures experts used equally

$$L_{aux} = \alpha \times \sum_{i=1}^{E} f_i \times P_i$$

where f_i = fraction routed to expert i

Benefits:

- $100\times$ parameters at $2\times$ cost
- Experts specialize (syntax, semantics, etc.)
- Conditional computation

MoE is how to scale beyond dense models

Worked Example: MoE Router Computation

Given: Input token x, 8 experts, top-2 selection

Step 1: Router computes logits

$$\text{logits} = W_g \cdot x = [2.1, 0.3, 1.8, 0.1, 3.2, 0.5, 1.1, 0.8]$$

Step 2: Select top-2
Top-2 experts: Expert 5 (3.2) and Expert 1 (2.1)

Step 3: Softmax over top-2 only

$$G_5 = \frac{\exp(3.2)}{\exp(3.2) + \exp(2.1)} = \frac{24.5}{32.6} = 0.75$$
$$G_1 = \frac{\exp(2.1)}{\exp(3.2) + \exp(2.1)} = \frac{8.17}{32.6} = 0.25$$

Step 4: Weighted sum

$$y = 0.75 \times E_5(x) + 0.25 \times E_1(x)$$

Sparsity: Only 2/8 experts activated = 75% reduction in computation!

Router learns which experts are relevant for which inputs

Switch Transformer: 1.6 Trillion Parameters

The Attention Complexity Problem

Efficient Transformers: Reducing Complexity

Reformer (LSH Attention):

- Locality-Sensitive Hashing
- Group similar queries/keys
- Attend only within groups
- Complexity: $O(n \log n)$

Linformer:

- Low-rank projection
- Project $n \times n$ to $n \times k$
- $k \ll n$ (e.g., 256)
- Complexity: $O(n)$

Linear Attention:

- Kernel trick
- Reorder operations
- Never compute $n \times n$ matrix
- Complexity: $O(n)$

When to Use:

- **Long documents**: $n > 4096$
- **Limited memory**: Consumer GPUs
- **Real-time**: Latency critical
- **Edge deployment**: Mobile, IoT

Multiple approaches to same problem - choose based on use case

Worked Example: Memory Savings from Efficient Attention

Given: Sequence length $n = 4096$, hidden dim $d = 512$

Standard Attention Memory:
Attention matrix: $n \times n = 4096 \times 4096 = 16.8M$ floats
Memory: $16.8M \times 4$ bytes $= 67$ MB per attention head
With 16 heads: $67 \times 16 = 1$ GB just for attention!

Linformer Memory ($k = 256$):
Projected matrix: $n \times k = 4096 \times 256 = 1M$ floats
Memory: $1M \times 4$ bytes $= 4$ MB per head
With 16 heads: $4 \times 16 = 64$ MB

Savings: $\frac{1GB}{64MB} = 16\times$ memory reduction!

Impact: Can process 4x longer sequences on same hardware

Efficient attention enables long-context applications

Technical Appendix

Deep Dive: GPT-3, MoE, Efficient Transformers

Appendix A1: GPT-3 Complete Specifications

Component	Small	Medium	Large	175B
Parameters	125M	350M	1.3B	175B
Layers	12	24	24	96
Hidden Size	768	1024	2048	12,288
Heads	12	16	16	96
Head Dimension	64	64	128	128
Context	2048	2048	2048	2048
Batch Size	0.5M	0.5M	1M	3.2M
Learning Rate	6e-4	3e-4	2.5e-4	1.2e-4
Performance				
LAMBADA (acc)	42.7	54.3	63.6	76.2
HellaSwag (acc)	43.6	54.7	67.4	78.9

Key Pattern: Consistent scaling across all metrics

Every metric improves smoothly with size

Appendix A2: GPT-3 Training Infrastructure

Hardware:

- 10,000+ NVIDIA V100 GPUs (32GB each)
- 285,000 CPU cores
- 400 Gbps network interconnect
- Custom Microsoft Azure infrastructure

Training Details:

- Distributed across thousands of nodes
- Model parallelism: Split model across GPUs
- Data parallelism: Different batches per GPU
- Pipeline parallelism: Layer-wise distribution
- Mixed precision training (FP16/FP32)

Challenges:

- Synchronization overhead
- Gradient accumulation across devices
- Fault tolerance (GPU failures during training)
- Checkpointing (model state = 350GB)

Training Time: Several months wall-clock time

Appendix A3: Few-Shot Prompting Best Practices

Prompt Engineering for GPT-3:

Zero-Shot:
`Translate to French: Hello` → Bonjour

One-Shot:
`English: Hello, French: Bonjour`
`English: Goodbye, French:` → Au revoir

Few-Shot (Optimal):
```
Translate English to French:
English: Hello / French: Bonjour
English: Goodbye / French: Au revoir
English: Thank you / French: Merci
English: Please / French: → S'il vous plaît
```

Best Practices:

- Use 3-10 examples (more doesn't always help)
- Examples should be diverse
- Format consistency critical
- Order matters (recency bias)
- Few-shot > fine-tuning for small datasets (¡ 100 examples)

Prompting is an art - small changes yield large effects

Appendix A4: GPT-3 API and Pricing

API Access:

- No model download (too large)
- API calls only (OpenAI servers)
- Multiple model sizes available

Pricing (2024):

Model	Input (per 1M tokens)	Output
GPT-3.5-turbo	$0.50	$1.50
GPT-4-turbo	$10.00	$30.00
GPT-4 (8K)	$30.00	$60.00

Rate Limits:

- Free tier: 3 requests/minute
- Paid tier: 3500 requests/minute
- Tokens per minute: 90K-2M depending on tier

Business Model: Amortize $4.6M training cost across millions of users

API pricing reflects compute cost and value delivered

Appendix A5: GPT-3 to GPT-4 Evolution

GPT-4 Improvements (2023):

- **Multimodal**: Images + text input
- **Larger context**: 32K tokens (16× GPT-3)
- **Better reasoning**: Chain-of-thought, mathematical
- **RLHF**: Reinforcement learning from human feedback
- **Parameters**: Rumored 1.7T (unconfirmed)

Performance Gains:

- Bar exam: 10th percentile → 90th percentile
- Coding: HumanEval 48% → 67%
- MMLU (general knowledge): 70% → 86%
- Reduced hallucinations by 40%

Architecture Speculation:

- Likely MoE (not confirmed)
- Multiple expert models combined
- Compute-optimal training (Chinchilla-informed)

GPT-4 shows continued scaling + better training

Appendix A6: MoE Router Network Mathematics

Router Architecture:

$$h = W_g x \in \mathbb{R}^E$$

where E = number of experts

Top-k Gating:

$$G(x) = \text{softmax}(\text{KeepTopK}(h, k))$$

KeepTopK sets all but top-k values to $-\infty$ before softmax

Sparse Gating Properties:

- Only k experts get non-zero weight
- Weights sum to 1 (softmax)
- Differentiable (can train with backprop)
- Gradient flows only through selected experts

Sparsity Benefit:
$k = 2$, $E = 128$: Activate $2/128 = 1.6\%$ of parameters
$100\times$ parameters at $2\times$ compute!

Sparsity is key - conditional computation based on input

Appendix A7: Load Balancing in MoE

Problem: Some experts get all traffic, others unused

Auxiliary Loss:

$$L_{aux} = \alpha \times \text{CV}(f_1, f_2, ..., f_E)^2$$

where CV = coefficient of variation, f_i = fraction routed to expert i

Penalizes imbalanced routing

Capacity Factor:
Limit tokens per expert:

$$\text{capacity}_i = \frac{\text{total tokens}}{E} \times \text{capacity factor}$$

Typical capacity factor = 1.25

Expert Choice Routing (alternative):
Experts choose tokens instead of tokens choosing experts
Better load balance, more stable training

Load balancing critical for effective expert utilization

Appendix A8: Switch Transformer Details

Google's Switch Transformer (2021):

- 1.6 trillion parameters (largest at time)
- Top-1 routing (simplest form of MoE)
- 2048 experts per layer
- Trained on C4 dataset (750GB)

Key Innovation: Simplified MoE

- Top-1 instead of top-2 (simpler)
- Expert capacity (limit tokens per expert)
- Smaller routers (reduced overhead)
- Better scaling than previous MoE

Results:

- 4× faster pre-training than T5-XXL (same quality)
- 7× faster fine-tuning
- Outperforms dense models at matched compute

Largest model trained demonstrates MoE viability

Appendix A9: Training Dynamics and Expert Specialization

What Do Experts Learn?
Empirical findings:

- Some experts specialize by syntax
- Some by semantics
- Some by domain (code, math, language)
- Specialization emerges during training

Training Challenges:

- **Mode collapse**: All traffic to few experts
- **Instability**: Router can oscillate
- **Expert imbalance**: Uneven utilization
- **Gradient noise**: Discrete routing not smooth

Solutions:

- Strong auxiliary losses
- Careful initialization
- Dropout on router
- Expert capacity constraints

MoE training trickier than dense - but rewards are huge

Appendix A10: MoE vs Dense - Complete Comparison

Aspect	Dense (GPT-3)	MoE (Switch)
Active Parameters	175B	10B
Total Parameters	175B	1600B
Sparsity	0%	99.4%
Memory (inference)	350GB	20GB
Training Time	Baseline	4× faster
Fine-tuning Time	Baseline	7× faster
Quality (matched compute)	Good	Better
Implementation	Simpler	Complex
Deployment	Standard GPUs	Needs coordination

When to Use MoE:

- Want massive capacity
- Have coordination infrastructure
- Training cost constrained
- Serving many requests (can batch)

When to Use Dense:

- Simpler deployment
- Low latency critical
- Small-scale serving

Appendix A11: Reformer and LSH Attention

Locality-Sensitive Hashing (LSH):

- Hash queries and keys to buckets
- Similar vectors $\rightarrow$ same bucket (high probability)
- Attend only within bucket

LSH Function:

$$h(x) = \text{argmax}([x \cdot r_1, x \cdot r_2, \ldots, x \cdot r_b])$$

where r_i are random projection vectors

Complexity:
Standard: $O(n^2)$
Reformer: $O(n \log n)$

Trade-offs:

- Much faster for $n > 4096$
- Approximate (misses some attention pairs)
- Enables sequences up to 64K tokens
- More complex implementation

Reformer enables long-context applications (books, long documents)

Appendix A12: Linformer Low-Rank Approximation

Key Idea: Attention matrix is approximately low-rank

Standard Attention:

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

where $QK^T \in \mathbb{R}^{n \times n}$

Linformer Modification:
Project keys and values to lower dimension k:

$$K' = KE_k \in \mathbb{R}^{n \times k}$$
$$V' = VF_k \in \mathbb{R}^{n \times k}$$

Then: $QK'^T \in \mathbb{R}^{n \times k}$ instead of $\mathbb{R}^{n \times n}$

Complexity:
Standard: $O(n^2 d)$
Linformer: $O(nkd)$ where $k = 256$ typical

For $n = 8192$, $k = 256$: $32\times$ speedup!

Simple idea, dramatic impact - linear complexity

Appendix A13: Linear Attention via Kernel Trick

Kernel Reformulation:
Standard: Attention$(Q, K, V) = \text{softmax}(QK^T)V$

Rewrite softmax as kernel: $\phi(q)^T \phi(k)$

Key Trick: Reorder operations

$$\text{Attention} = \phi(Q)(\phi(K)^T V)$$

Compute $\phi(K)^T V$ first! This is $k \times d$ (small)

Complexity:

- Standard: $(n \times n) \times (n \times d) = O(n^2 d)$
- Linear: $(n \times k) \times (k \times d) = O(nkd)$
- With $k = d$: $O(nd^2)$ - linear in n!

Approximation Quality:
Not exact, but empirically good
Works well for long sequences

Kernel trick enables true linear attention

Appendix A14: FlashAttention - GPU Optimization

Problem: Standard attention is IO-bound, not compute-bound

GPU memory hierarchy:

- SRAM (on-chip): Fast but tiny (20MB)
- HBM (GPU RAM): Slow but large (40GB)

Standard Attention IO:

1. Load Q, K from HBM
2. Compute $S = QK^T$ (write to HBM)
3. Load S from HBM
4. Apply softmax (write to HBM)
5. Load P, V from HBM
6. Compute PV (write to HBM)

Multiple slow HBM reads/writes!

FlashAttention Optimization:

- Fuse operations (compute in SRAM)
- Tiling (process in blocks)
- Never materialize full $n \times n$ matrix
- 2-4$\times$ faster, less memory

Algorithm innovation - same math, better hardware utilization

Appendix A15: Choosing Efficient Transformer Variant

Variant	Complexity	Exact	Max n	Use Case
Standard	$O(n^2)$	Yes	2K-4K	Default
Reformer	$O(n \log n)$	No	64K	Long docs
Linformer	$O(n)$	No	32K	Fast training
Linear Attn	$O(n)$	No	16K	Real-time
FlashAttention	$O(n^2)$	Yes	8K	GPU-optimized
Best for:				
Books (100K+)	Reformer			
Articles (8-16K)	Linformer			
Production (2-4K)	FlashAttn			
Mobile/Edge	Linear Attn			

Recommendation:

- Start with FlashAttention (better standard)
- Use Linformer if need $n > 4K$
- Reformer for extreme lengths ($n > 32K$)
- Linear attention for edge deployment

Choose based on sequence length and deployment constraints