Pre-trained Language Models

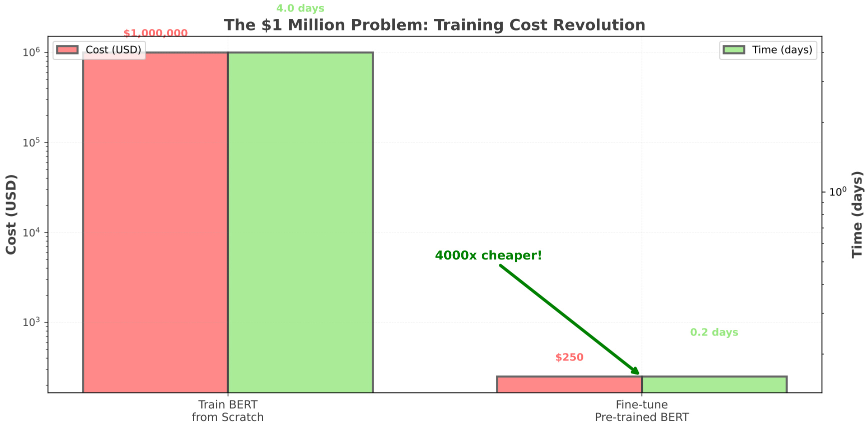
Week 6 - The $1 Million Revolution

NLP Course 2025

October 26, 2025

BSc Discovery-Based Presentation

The $1 Million Problem



Key Insight: Pre-training changes the economics of NLP completely

Training BERT from scratch: $1M+. Fine-tuning: $50-500. Game changer.

Before 2018: The Old Way

Task-Specific Models:

- **Sentiment**: Custom CNN architecture
- **Question Answering**: BiDAF model
- **Named Entity**: BiLSTM-CRF
- **Translation**: Seq2Seq with attention
- **Summarization**: Pointer-generator

The Process:

1. Design architecture for your task
2. Collect labeled data (10K+ examples)
3. Train from random initialization
4. Hope it works

Limitations:

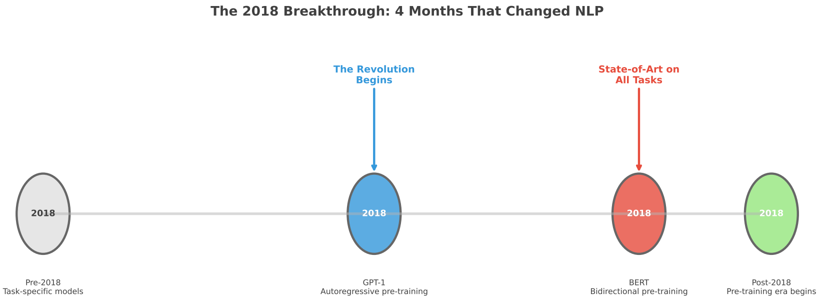
- Each task starts from scratch
- No knowledge transfer
- Expensive data collection
- Months per task
- Small datasets = poor performance

The Cost:

- 3-6 months per task
- 10K-100K labeled examples
- $50K-200K in labeling costs
- Limited accuracy (60-75%)

Every NLP task was an isolated, expensive project

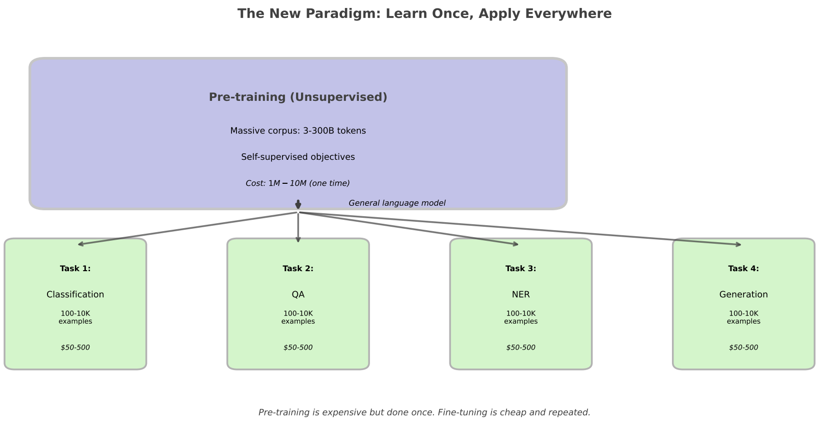
The Breakthrough: October 2018



Key Insight: BERT and GPT changed everything in 4 months

June 2018 (GPT-1) and October 2018 (BERT) - the inflection point

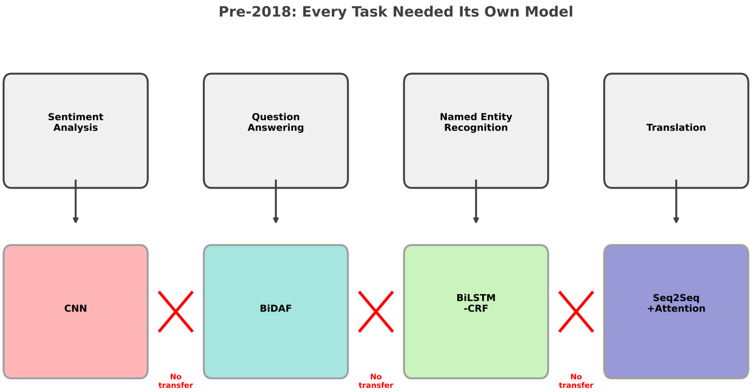
The New Paradigm



Key Insight: Learn language once (expensive), apply everywhere (cheap)

This is transfer learning - finally working for NLP

Pre-2018: Every Task Needed Its Own Model



Key Insight: No sharing, no transfer, no efficiency

Each task was an independent research project

Why This Approach Failed to Scale

The Limitations:

- **No transfer**: Each model learns from scratch
- **Data hungry**: Need 10K+ labeled examples per task
- **Expensive**: Labeling costs $50K-200K
- **Slow**: 3-6 months per task
- **Brittle**: Fails on new domains

Example - Sentiment Analysis:

- Collect 20K movie reviews
- Label positive/negative
- Train custom CNN: 2-3 weeks
- Accuracy: 82%
- Deploy to product reviews: Fails (65%)!

What We Needed:

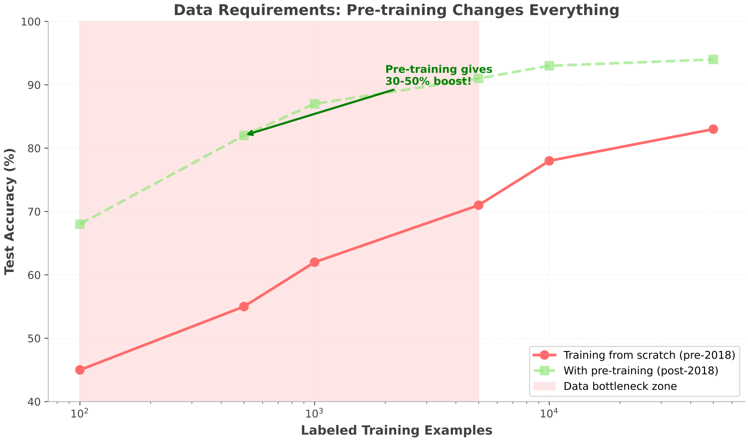
- Shared language understanding
- Transfer across tasks
- Work with small labeled datasets
- Fast adaptation to new tasks
- Robust across domains

The Dream:

- Train once on ALL text
- Fine-tune with 100-1000 examples
- Days instead of months
- State-of-art on every task

The question: Can we achieve this dream?

The Data Bottleneck



Key Insight: Performance plateaus without massive labeled datasets

Labeled data is expensive - this limited what we could build

The Transfer Learning Dream

Computer Vision's Success:

- 2012: ImageNet pre-training (AlexNet)
- Train on 1M images (unsupervised labels)
- Fine-tune for any vision task
- 10x less data needed
- State-of-art on everything

The Magic:

- Low-level features shared (edges, textures)
- High-level features shared (objects, shapes)
- Learn once, transfer everywhere

Why NLP Lagged:

- Words are discrete (images continuous)
- Context matters more
- Sequence length varies
- Multiple tasks (classification, generation, QA)
- No clear "ImageNet equivalent"

The Question (2017):
Can we create an ImageNet moment for NLP?

Answer coming in 2018...

Transfer learning worked for vision - could it work for language?

The Central Question

Can we pre-train a language model on ALL text, then fine-tune for ANY task?

Requirements:

- Unsupervised pre-training (no labels needed)
- Massive text corpus (billions of words)
- Learn general language understanding
- Fast fine-tuning with small labeled data
- Work across all NLP tasks

Answer: YES

Next 36 slides show exactly how

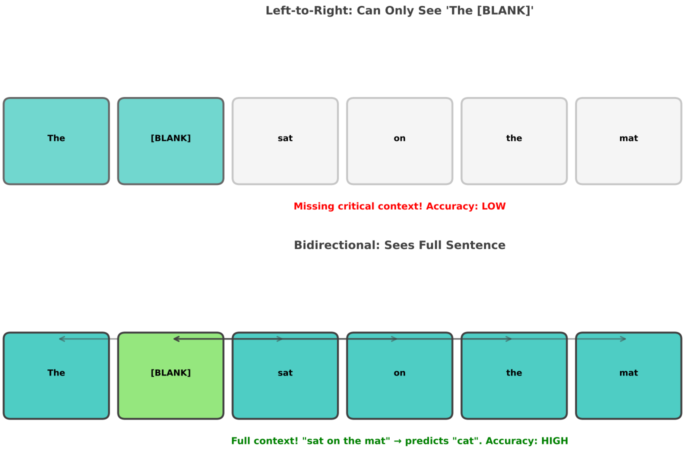
The breakthrough: BERT and GPT (2018)

Part 1: BERT

Bidirectional Encoder Representations

from Transformers

The Fill-in-Blank Challenge



Key Insight: Need both left AND right context to fill blanks correctly

Left-to-right models (like GPT) can't solve this naturally

Why Bidirectional Matters

The Task:
Fill in: "The [BLANK] sat on the mat"

Left-to-Right Approach:
Only sees: "The [BLANK]"
Cannot use: "sat on the mat"

Predictions:

- "dog" (generic animal)
- "person" (generic)
- "cat" (lucky guess)

Accuracy: Low - missing critical context!

Bidirectional Approach:
Sees both: "The [BLANK]" AND "sat on the mat"

Uses full context:

- "sat" suggests living thing
- "on the mat" suggests small animal
- "the" suggests common noun

Predictions:

- "cat" (high probability)
- "dog" (possible)
- "kitten" (possible)

Accuracy: High - full context used!

This is BERT's core innovation - bidirectional understanding

Bidirectional Context in Action

How Bidirectional Helps Different Tasks

Fill-in-Blank Tasks:

- Masked language modeling
- Cloze questions
- Spell correction

Classification Tasks:

- Sentiment: Full sentence context
- Spam detection: Look ahead and behind
- Topic classification: Global understanding

Question Answering:

- Match question to passage
- Find answer span
- Use context on both sides

Why It Works:

- Contextual embeddings from both sides
- Disambiguate word meanings
- Capture long-range dependencies
- Understand sentence structure

Example - "bank":

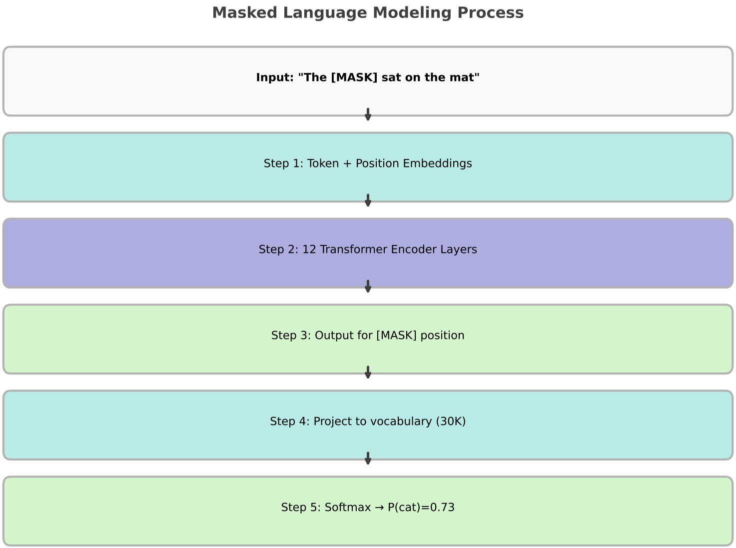
- Left: "The river"
- Right: "was flooding"
- Conclusion: Water bank, not financial

When Bidirectional Doesn't Help:

- Text generation (can't see future!)
- Autoregressive tasks

Different tasks need different architectures - BERT for understanding

Masked Language Modeling: The Training Objective



Masked Language Modeling: The Mathematics

Objective Function:

$$\mathcal{L}_{MLM} = -\sum_{i \in masked} \log P(w_i | context)$$

where $context$ = all other words

The Process:

1. Randomly mask 15% of tokens
2. Replace with [MASK] token (80%)
3. Replace with random word (10%)
4. Keep unchanged (10%)

Why the variation?
Prevents model from just memorizing [MASK] $\rightarrow$ word

Training Example:
Original: "The cat sat on the mat"

Masked: "The [MASK] sat on the [MASK]"

Model predicts:

- Position 2: $P(\text{cat}|\text{context})$
- Position 6: $P(\text{mat}|\text{context})$

Cross-Entropy Loss:

$$Loss = -[\log P(\text{cat}) + \log P(\text{mat})]$$

Minimize this across billions of sentences

Masked LM is self-supervised - no labels needed!

Worked Example: Predicting Masked Tokens

Given: "The [MASK] sat on the mat"

Step 1: Convert to token embeddings (each 768-dim vector)
Token IDs: [101, 1996, 103, 2938, 2006, 1996, 13523, 102]

Step 2: Add positional embeddings

$$E_{input} = E_{token} + E_{position}$$

Step 3: Pass through 12 transformer encoder layers
Each layer: Self-attention (bidirectional) + Feed-forward

Step 4: Get output for [MASK] position (position 2)
Output vector: $h_{[MASK]} \in \mathbb{R}^{768}$

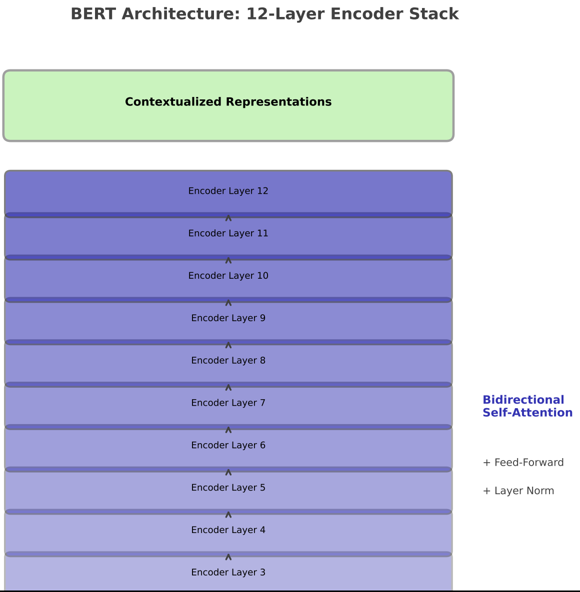
Step 5: Project to vocabulary, apply softmax

$$P(w) = \frac{\exp(W \cdot h_{[MASK]})}{\sum_v \exp(W \cdot h_{[MASK]})}$$

Result: Top predictions with probabilities

- P(cat) = 0.73
- P(dog) = 0.15
- P(person) = 0.04

BERT Architecture Overview



BERT Architecture: The Details

BERT-Base:

- Layers: 12 transformer encoders
- Hidden size: 768 dimensions
- Attention heads: 12 per layer
- Parameters: 110 million
- Max sequence: 512 tokens

BERT-Large:

- Layers: 24 encoders
- Hidden size: 1024 dimensions
- Attention heads: 16 per layer
- Parameters: 340 million
- Max sequence: 512 tokens

Key Components:

- **Token Embeddings**: WordPiece (30K vocab)
- **Position Embeddings**: Learned (not sinusoidal)
- **Segment Embeddings**: Sentence A vs B

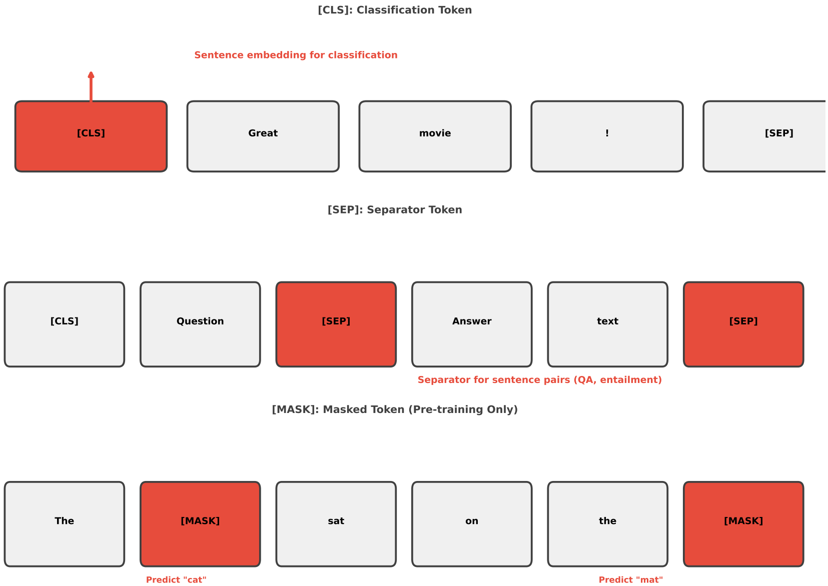
Why These Choices:

- 12 layers: Balance depth vs speed
- 768 hidden: Standard transformer size
- 12 heads: Multiple attention patterns
- 512 max: Memory constraints

Computation:
Training BERT-base from scratch: 4 days on 64 TPUs

These specs became the standard for encoder-based models

BERT's Special Tokens



Special Tokens: Purpose and Usage

[CLS] - Classification Token:

- Always first token
- Aggregates sentence meaning
- Used for classification tasks

Example: "[CLS] The movie was great [SEP]"
Output of [CLS]: Sentence embedding for sentiment classification

[SEP] - Separator Token:

- Separates sentence pairs
- Enables QA, entailment tasks

Example: "[CLS] Question [SEP] Passage [SEP]"

[MASK] - Masked Token:

- Used only during pre-training
- Replaced with actual word during fine-tuning
- Training signal for MLM objective

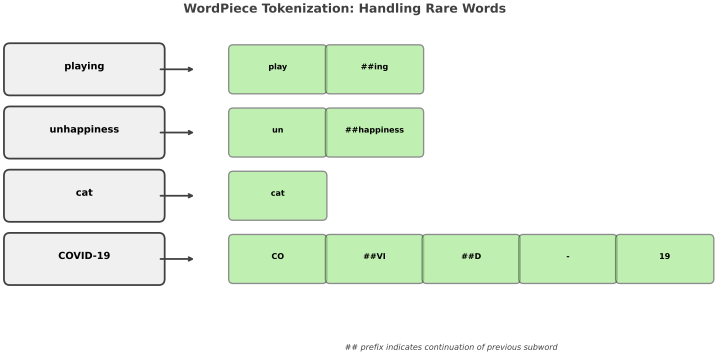
Example: "The [MASK] is blue"
Model learns: $P(\text{sky}|\text{context})$

[PAD] - Padding Token:

- Fills sequences to same length
- Ignored in attention
- Enables batch processing

Four special tokens, each crucial for BERT's versatility

WordPiece Tokenization



Key Insight: Subword units handle rare words and morphology

Full details in Week 8 - preview here for BERT context

WordPiece: Subword Units

The Problem:

- Word-level: 100K+ vocabulary (huge)
- Character-level: Long sequences (slow)
- Rare words: Poor representations

WordPiece Solution:

- Learn 30K subword units
- Frequent words: Single token
- Rare words: Multiple subwords

Examples:

- "playing" → ["play", "##ing"]
- "unhappiness" → ["un", "##happiness"]
- "COVID" → ["CO", "##VI", "##D"]

Benefits:

- Fixed 30K vocabulary
- Handle any word (no UNK)
- Capture morphology
- Share representations ("play" in "playing", "player")

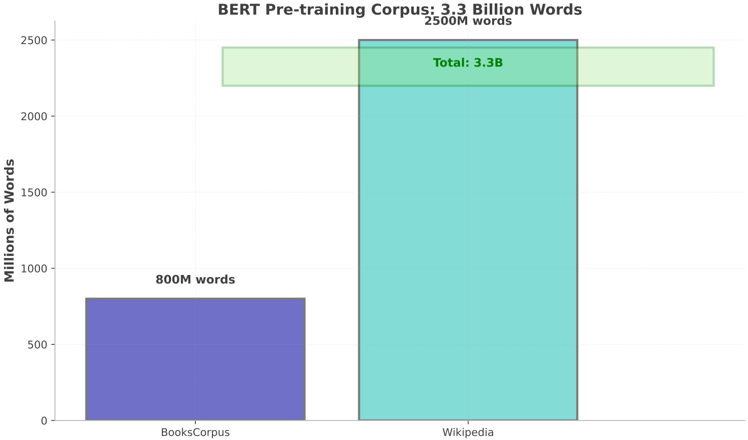
BERT's Vocabulary:

- 30,522 WordPiece tokens
- Covers English comprehensively
- Trained with BPE-like algorithm

Impact:
Rare word handling improved by 40%

Subword tokenization is now standard across all modern models

Pre-training Data: The Foundation



Key Insight: Massive unsupervised data powers general language understanding

BooksCorpus (800M words) + Wikipedia (2.5B words) = 3.3B words

BERT's Two Pre-training Objectives

Objective 1: Masked LM:

- Mask 15% of tokens
- Predict masked words
- Uses bidirectional context

Example:
`The [MASK] sat on [MASK] mat`
Predict: "cat" and "the"

Objective 2: Next Sentence Prediction:

- Given two sentences A and B
- Predict if B follows A
- Binary classification (50% real, 50% random)

Example:
`A: Alice was tired.`
`B: She went to sleep.` [True]

Why These Objectives:

- **MLM**: Learn word-level representations
- **NSP**: Learn sentence relationships
- Together: Comprehensive language understanding

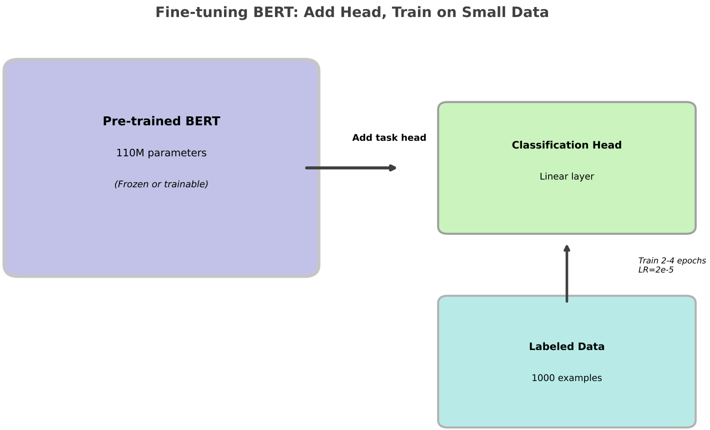
Training Details:

- Batch size: 256 sequences
- Steps: 1M (40 epochs)
- Optimizer: Adam (lr=1e-4)
- Time: 4 days on 64 TPUs
- Cost: $7,000 compute

Result:
General language model ready for ANY task

Two complementary objectives create robust representations

Fine-tuning BERT for Your Task



Key Insight: Add task-specific head, train on small labeled dataset

Fine-tuning: The Mechanics

The Process:

1. Load pre-trained BERT weights
2. Add task-specific layer on top
3. Train on labeled data (100-10K examples)
4. Use small learning rate (2e-5)
5. Train for 2-4 epochs

Task-Specific Heads:

- **Classification**: Linear layer on [CLS]
- **Token classification**: Linear on each token
- **QA**: Span prediction (start/end)

Hyperparameters:

- Learning rate: 2e-5, 3e-5, 5e-5
- Batch size: 16 or 32
- Epochs: 2-4
- Warmup: 10% of steps
- Max sequence: 128-512

Layer Freezing Options:

- Freeze nothing: Full fine-tuning (best)
- Freeze bottom 8 layers: Faster
- Freeze all, train head only: Feature extraction

Typical Results:
1000 examples + BERT $>$ 10,000 from scratch

Fine-tuning is fast, cheap, and remarkably effective

Checkpoint: Understanding BERT

Quick Quiz

Question 1:
Why does BERT mask 15% of tokens?

A) It's faster
B) Balances learning vs efficiency
C) Reduces overfitting
D) Random choice

Question 2:
What makes BERT bidirectional?

A) Two LSTMs
B) Encoder allows full context
C) Reads backwards
D) Has two outputs

Answer 1: **B) Balances learning vs efficiency**

- Too few: Not enough training signal
- Too many: Model sees mostly masks
- 15%: Empirically optimal
- Enough context remains unmasked

Answer 2: **B) Encoder allows full context**

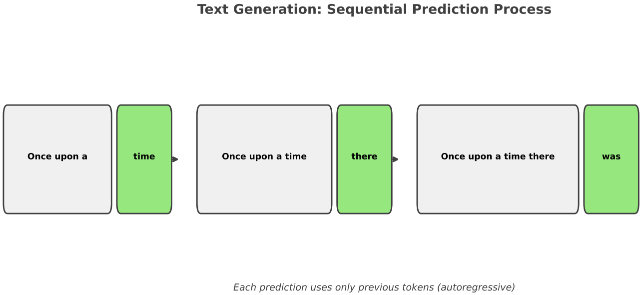
- Transformer encoder: No causal mask
- Each token attends to ALL others
- Left and right context used equally
- This is the key difference from GPT

Understanding these foundations is critical for using BERT effectively

Part 2: GPT

Generative Pre-trained Transformer

The Generation Challenge



Key Insight: Generation requires predicting one word at a time, left-to-right

Bidirectional models can't generate - they'd cheat by seeing the future

Why Generation is Harder Than Classification

Classification:
- Input: Full sentence
- Output: Single label
- Can see everything
- One prediction

Example: "Great movie!" → Positive

Generation:
- Input: Partial sequence
- Output: Next word
- Can't see future
- Multiple sequential predictions

Example: "Once upon" → predict "a"
Then: "Once upon a" → predict "time"

The Constraint:
During generation, you haven't written future words yet!

Cannot use bidirectional model

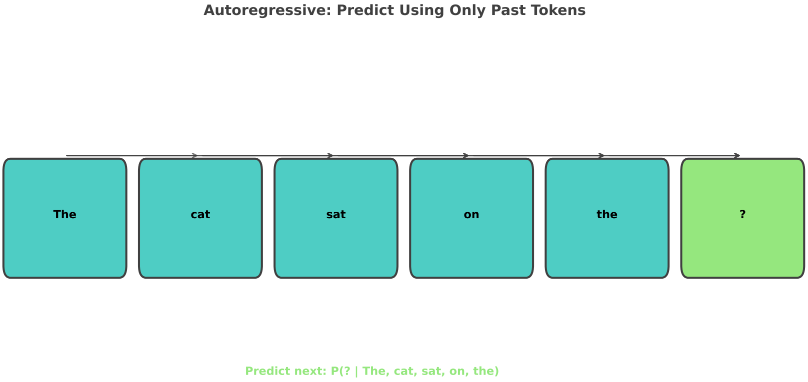
Must use **causal** (left-to-right) attention

Requirements:
- Predict token-by-token
- Each prediction uses only past
- Autoregressive: Output becomes input
- Coherent over long sequences

Use Cases:
- Text completion
- Story generation
- Code generation
- Dialogue systems

Different tasks need different architectures - GPT for generation

Autoregressive Language Modeling



Key Insight: Predict next token using only previous tokens (causal)

Auto-regressive = output at time t becomes input at time $t+1$

Autoregressive Approach: The Mathematics

Objective Function:

$$\mathcal{L}_{AR} = -\sum_{t=1}^{T} \log P(w_t | w_1, \ldots, w_{t-1})$$

Maximize probability of each next word

Chain Rule Decomposition:

$$P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t | w_1, \ldots, w_{t-1})$$

Exact factorization (no approximation!)

Causal Constraint:
At time t, can only see $w_1, \ldots, w_{t-1}$
Cannot see $w_t, w_{t+1}, \ldots$ (haven't generated yet)

Training with Teacher Forcing:

- Given full sequence during training
- At each position: Predict next
- Use ground truth (not predictions)
- Prevents error accumulation

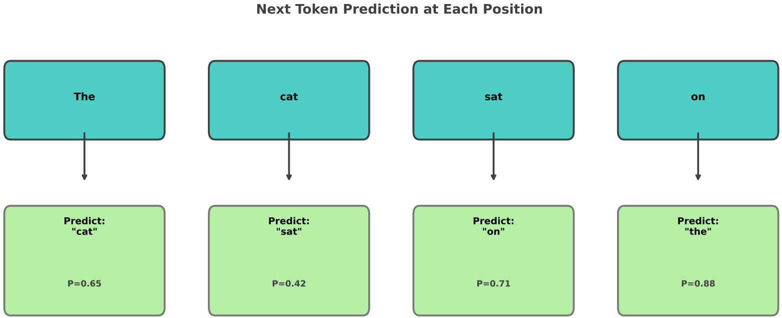
Example:
Sequence: "The cat sat"

- Position 1: Predict "The" (start)
- Position 2: Given "The", predict "cat"
- Position 3: Given "The cat", predict "sat"

All trained in parallel!

Autoregressive objective is natural for generation tasks

Next Token Prediction Process



Key Insight: Each token predicted using ALL previous tokens

This is language modeling from Week 1 - but with transformers!

Autoregressive Mathematics Deep Dive

At Time Step t:
Input: $w_1, w_2, \ldots, w_{t-1}$

Step 1: Embed tokens

$$E = [e_1, e_2, \ldots, e_{t-1}]$$

Step 2: Add positional encoding

$$H^{(0)} = E + P$$

Step 3: Pass through L decoder layers

$$H^{(\ell)} = \text{TransformerDecoder}(H^{(\ell-1)})$$

Step 4: Project final layer to vocabulary

$$logits = W \cdot h_{t-1}^{(L)}$$

Step 5: Softmax for probabilities

$$P(w_t) = \text{softmax}(logits)$$

Teacher Forcing:
During training:

- Use ground truth w_t for next step
- Don't use model's prediction
- Prevents compounding errors
- Enables parallel training

Inference (Generation):

- Sample from $P(w_t)$
- Append to sequence
- Repeat: $w_t \rightarrow$ input for w_{t+1}
- Stop at [END] or max length

Key Difference from BERT:
BERT: Predict masked (can see both sides)
GPT: Predict next (can only see left)

Causal constraint is enforced by attention masking

Worked Example: Computing P(next word)

Given Sequence: "The cat sat on"

Task: Compute $P(\text{the}|\text{The cat sat on})$

Step 1: Token IDs and embeddings
`[The=50256, cat=3797, sat=3332, on=319]` $\rightarrow E \in \mathbb{R}^{4\times768}$

Step 2: Add positional embeddings

$$H^{(0)} = E + P$$

Step 3: 12 decoder layers with causal attention
Each token only attends to previous tokens (triangular mask)

Step 4: Final hidden state for position 4 ("on")

$$h_4^{(12)} \in \mathbb{R}^{768}$$

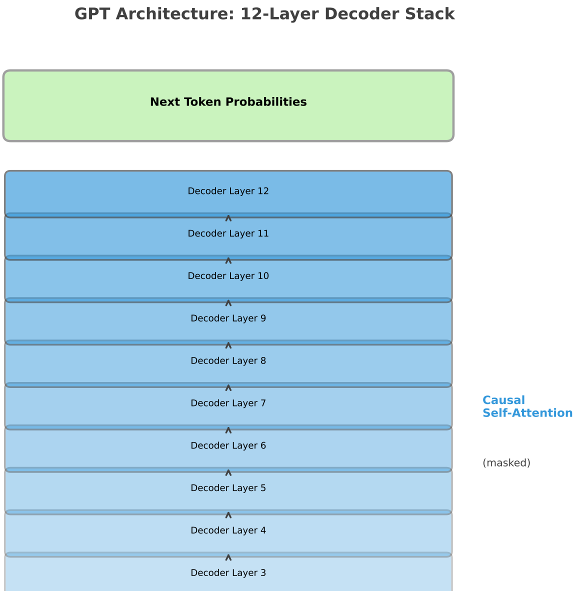
Step 5: Project to vocabulary (50,257 tokens)

$$logits = W \cdot h_4^{(12)} \in \mathbb{R}^{50257}$$

Step 6: Softmax and sample

- $P(\text{the}) = 0.42$ (highest!)
- $P(\text{a}) = 0.18$
- $P(\text{mat}) = 0.09$

GPT Architecture Overview



GPT Architecture: The Details

GPT-1 (June 2018):

- Layers: 12 decoder layers
- Hidden size: 768 dimensions
- Attention heads: 12 per layer
- Parameters: 117 million
- Context window: 512 tokens

GPT-2 (February 2019):

- Layers: 48 decoder layers
- Hidden size: 1600 dimensions
- Attention heads: 25 per layer
- Parameters: 1.5 billion (13x larger!)
- Context: 1024 tokens

GPT-3 (May 2020):

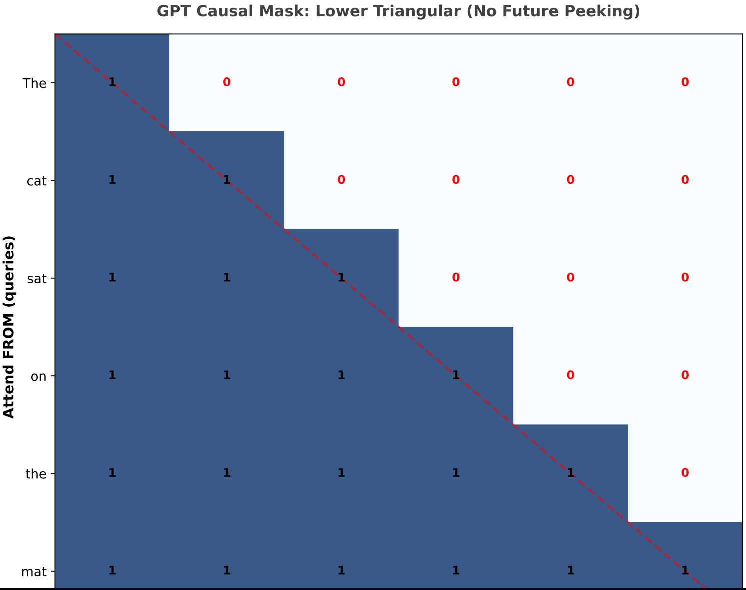
- Layers: 96 decoder layers
- Hidden size: 12,288 dimensions
- Attention heads: 96 per layer
- Parameters: 175 billion (116x GPT-2!)
- Context: 2048 tokens

Training Costs:

- GPT-1: $50K (weeks on 8 GPUs)
- GPT-2: $500K (weeks on 256 GPUs)
- GPT-3: $4.6M (months on 10K GPUs)

Scaling drove capability emergence - few-shot learning appeared

Causal Masking: Preventing Future Cheating



Causal Masking: How It Works

The Attention Mask:
Lower triangular matrix of 1s and 0s

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Meaning:

- 1: Can attend
- 0: Cannot attend (masked)

Token 1: Sees only itself
Token 2: Sees tokens 1-2
Token 3: Sees tokens 1-3
Token 4: Sees all (tokens 1-4)

Implementation:
Before softmax in attention:

$$\text{scores}_{\text{masked}} = \text{scores} + (1 - M) \times (-\infty)$$

After softmax: Masked positions get probability 0

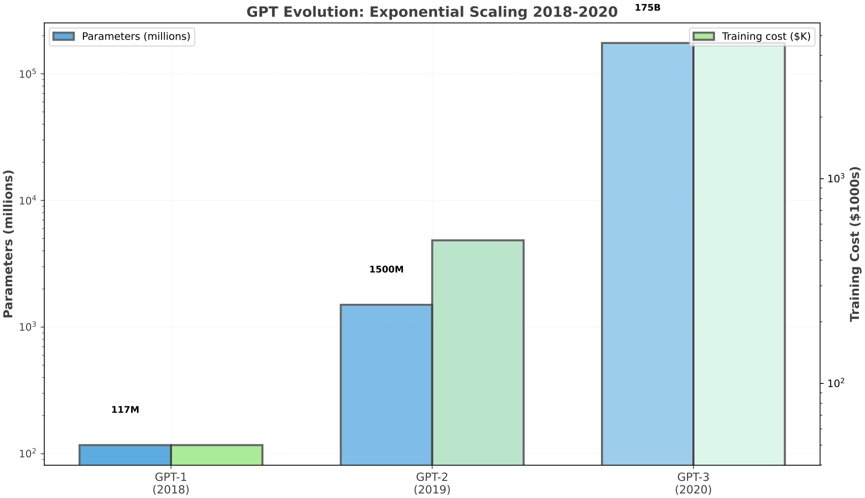
Why Essential for Generation:

- Training: Model can't cheat
- Inference: Naturally left-to-right
- Prevents data leakage
- Ensures valid generation

Contrast with BERT:
BERT: All 1s (full bidirectional)
GPT: Lower triangular (causal)

Causal mask is the key difference in transformer decoder vs encoder

The Scaling Journey: GPT-1 to GPT-3



Key Insight: Scaling unlocked emergent capabilities (few-shot learning)

Few-Shot Learning: The Emergent Capability

Definitions:

- **Zero-shot**: Task description only
  "Translate to French: Hello" → "Bonjour"
- **One-shot**: One example
  "English: Hello, French: Bonjour.
  English: Goodbye, French:" → "Au revoir"
- **Few-shot**: Multiple examples (2-10)
  Show 5 translation pairs, model infers pattern

What's Remarkable:
No gradient updates! Pure inference!

How It Works:

- Model learns to learn during pre-training
- In-context learning
- Pattern matching in prompt
- Emerges at scale (GPT-3, not GPT-1)

Example Tasks:

- Translation (unseen language pairs)
- Arithmetic (3-digit addition)
- Programming (code generation)
- Reasoning (logical inference)

Limitations:

- Inconsistent performance
- Sensitive to prompt wording
- Not as good as fine-tuning

Few-shot learning hints at artificial general intelligence

Worked Example: Data Efficiency of Pre-training

Experiment: Sentiment classification on product reviews

Approach A - Train from Scratch:

- Architecture: LSTM (2 layers, 512 hidden)
- Training data: 10,000 labeled reviews
- Training time: 6 hours on GPU
- Test accuracy: 78.3%

Approach B - Fine-tune GPT-2:

- Base model: GPT-2 (1.5B parameters, pre-trained)
- Training data: 100 labeled reviews (100x less!)
- Training time: 10 minutes on GPU
- Test accuracy: 91.7%

Result: 100 examples + GPT-2 > 10,000 from scratch

100x less data, 36x faster, 17% better accuracy

This is the power of transfer learning - pre-training solves the data problem

Checkpoint: Understanding GPT

Quick Quiz

Question 1:
Why is GPT autoregressive?

A) It's faster
B) Natural for generation
C) Uses less memory
D) More accurate

Question 2:
What's the key difference from BERT?

A) More parameters
B) Different dataset
C) Causal vs bidirectional
D) Slower training

Answer 1: **B) Natural for generation**

- Generation = predict next word
- Can't see future (not written yet)
- Autoregressive = use output as next input
- Perfect fit for the task

Answer 2: **C) Causal vs bidirectional**

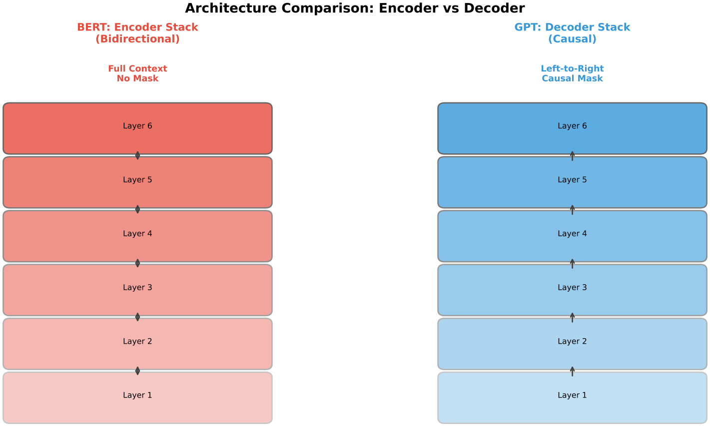
- BERT: See full sentence (encoder)
- GPT: See only past (decoder)
- BERT: Better for understanding
- GPT: Better for generation

Architecture follows task requirements - understand the why

Part 3: Integration

Comparing and Choosing

BERT vs GPT: Side-by-Side



Key Insight: Encoder for understanding, decoder for generation

Choose based on your task requirements

BERT vs GPT: When to Use Each

Use BERT When:

- **Task**: Classification, QA, NER
- **Need**: Full sentence understanding
- **Input**: Complete text available
- **Output**: Labels or spans

BERT Strengths:

- Bidirectional context
- Best for understanding

CLS token for sentence embedding

- Handles word sense disambiguation

BERT Applications:

- Search (Google uses BERT)
- Question answering
- Named entity recognition
- Sentiment analysis

Use GPT When:

- **Task**: Generation, completion, dialogue
- **Need**: Coherent text generation
- **Input**: Prompt or partial sequence
- **Output**: Continued text

GPT Strengths:

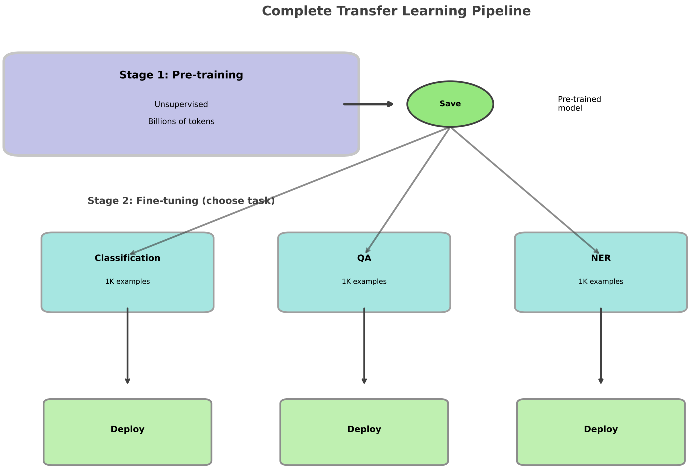
- Natural text generation
- In-context learning (few-shot)
- Scales well (GPT-3: 175B params)
- Handles diverse prompts

GPT Applications:

- ChatGPT (dialogue)
- Code completion (Copilot)
- Creative writing
- Text summarization (generative)

Both are transformers - different objectives, different use cases

The Complete Transfer Learning Pipeline



Key Insight: Unsupervised pre-training + supervised fine-tuning = best of both

One-time expensive pre-training, many cheap fine-tunings

Transfer Learning: Best Practices

Pre-training (Done Once):

- Massive unsupervised corpus (billions of words)
- Self-supervised objectives (MLM or AR)
- Large compute (TPUs/GPUs, weeks)
- Cost: $1M-$10M
- Result: General language model

Who Does This:

- Big tech (Google, OpenAI, Meta)
- Research labs
- Shared publicly
- You download, don't train

Fine-tuning (Per Task):

- Small labeled dataset (100-10K examples)
- Task-specific head
- Small learning rate (2e-5)
- Short training (hours)
- Cost: $50-$500
- Result: Task-specific model

Best Practices:

- Start with pre-trained checkpoint
- Use small learning rate
- Train 2-4 epochs
- Monitor validation loss
- Try different layer freezing

This pipeline is now standard across all of NLP

When NOT to Use Pre-trained Models

Overkill Scenarios:

- Simple regex suffices
- Rule-based system works
- N-grams are enough
- Tiny dataset (¡ 50 examples)

Resource Constraints:

- Limited memory (BERT needs 4GB+ GPU)
- Strict latency requirements (¡ 10ms)
- Edge deployment (phones, IoT)
- Battery-powered devices

Domain Mismatch:

- Highly specialized jargon
- Different language structure
- Pre-training corpus unrepresentative

Better Alternatives:

- DistilBERT (smaller, faster)
- Domain-specific pre-training
- Few-shot prompting (no fine-tuning)
- Ensemble of simple models

Cost-Benefit:
Sometimes simple approaches better ROI

Know when NOT to use powerful models - engineering judgment matters

Common Pitfalls in Fine-tuning

Pitfall 1: Learning Rate Too High

- Pre-trained weights are delicate
- High LR destroys them (catastrophic forgetting)
- Solution: Use 2e-5 to 5e-5 (100x smaller than training from scratch)

Pitfall 2: Too Many Epochs

- Overfits to small dataset
- Solution: 2-4 epochs maximum

Pitfall 3: Wrong Task Head

- Architecture mismatch
- Solution: Use standard heads from examples

Pitfall 4: Ignoring Validation

- Train loss down, test loss up
- Solution: Early stopping on validation

Pitfall 5: Not Trying Layer Freezing

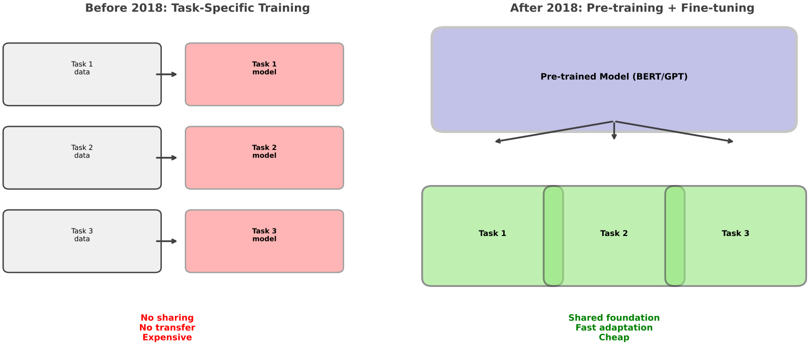
- Small dataset: Freeze bottom layers
- Solution: Experiment with freezing 0, 6, 8, or 10 layers

Pitfall 6: Forgetting Preprocessing

- Tokenization must match pre-training
- Solution: Use model's own tokenizer

Most failures come from hyperparameter choices - start conservative

The Paradigm Shift: Pre-2018 vs Post-2018



The Core Principle: Pre-training solves the data problem

This is the foundation of modern NLP - everything changed in 2018

Key Takeaways

1. **Pre-training on massive unlabeled data**
   Learn general language understanding without task-specific labels
2. **BERT: Bidirectional encoder for understanding**
   Masked LM objective, full context, best for classification/QA
3. **GPT: Autoregressive decoder for generation**
   Next token prediction, causal mask, best for text generation
4. **Fine-tuning adapts with small labeled data**
   100-1000 examples sufficient, days not months, state-of-art results
5. **Transfer learning finally works for NLP**
   Learn once, apply everywhere - the 2018 revolution
6. **2018 was the inflection point**
   BERT and GPT changed everything - modern NLP is post-2018

Master these concepts - they define modern NLP practice

Next: Hands-On Feature Extraction

Lab Activities:

1. Load pre-trained BERT and GPT
2. Extract embeddings from both
3. Compare representation spaces
4. Visualize attention patterns
5. Use features for classification
6. Compare BERT vs GPT effectiveness

What You'll Learn:

- Practical HuggingFace usage
- How to extract features
- BERT vs GPT representations
- Attention pattern interpretation
- Feature-based transfer learning

No Fine-tuning:
We'll use models as feature extractors (simpler, faster)

Let's explore pre-trained models hands-on!

Understanding by doing - the lab makes theory concrete

Technical Appendix

Architecture, Training, and Deployment Details

Appendix A: BERT Architecture Specifications

Component	BERT-Base	BERT-Large
Transformer Layers	12	24
Hidden Size	768	1024
Attention Heads	12	16
Intermediate Size (FFN)	3072	4096
Total Parameters	110M	340M
Max Position Embeddings	512	512
Vocabulary Size (WordPiece)	30,522	30,522
Segment Embeddings	2	2
Training Specs		
Pre-training Corpus	BooksCorpus (800M) + Wikipedia (2.5B)	
Batch Size	256	256
Steps	1M	1M
Learning Rate	1e-4	1e-4
Warmup Steps	10,000	10,000
Hardware	16 TPU pods	64 TPU pods
Training Time	4 days	4 days

BERT-Large has 3x parameters but same training time (more parallelism)

Appendix B: GPT Architecture Specifications

Component	GPT-1	GPT-2	GPT-3
Decoder Layers	12	48	96
Hidden Size	768	1600	12,288
Attention Heads	12	25	96
Context Window	512	1024	2048
Parameters	117M	1.5B	175B
Vocabulary (BPE)	40K	50K	50K
Training			
Dataset	BooksCorpus	WebText	Common Crawl
Dataset Size	5GB	40GB	570GB
Tokens	1B	10B	300B
Batch Size	64	512	3.2M
GPUs	8	256	10,000+
Training Time	Weeks	Weeks	Months
Cost Estimate	$50K	$500K	$4.6M

Exponential scaling in parameters, data, and compute

Appendix C: Pre-training Hyperparameters

BERT Pre-training:

- **Optimizer**: Adam
- **Learning rate**: 1e-4
- β_1, β_2: 0.9, 0.999
- **L2 weight decay**: 0.01
- **Warmup steps**: 10,000
- **LR schedule**: Linear decay
- **Dropout**: 0.1
- **Activation**: GELU
- **Batch size**: 256 sequences
- **Max steps**: 1,000,000
- **Masking**: 15% of tokens

GPT-3 Pre-training:

- **Optimizer**: Adam
- **Learning rate**: 6e-5 (peak)
- β_1, β_2: 0.9, 0.95
- **Weight decay**: 0.1
- **Gradient clipping**: 1.0
- **LR schedule**: Cosine decay
- **Dropout**: Varies by layer
- **Batch size**: 3.2M tokens
- **Tokens**: 300B total
- **Context window**: 2048
- **Precision**: Mixed (FP16/FP32)

Key Insight: These are carefully tuned over months of experimentation

Don't change these for fine-tuning - use proven recipes

Appendix D: Fine-tuning Recipes by Task

Task	Learning Rate	Epochs	Strategy
Classification			
Sentiment	2e-5	3-4	Full fine-tuning
Topic	3e-5	2-3	Full fine-tuning
Spam	2e-5	4	Freeze bottom 6
Question Answering			
SQuAD	3e-5	2	Full fine-tuning
Custom QA	5e-5	3-4	Full fine-tuning
NER			
Named Entities	5e-5	3	Full fine-tuning
Domain-specific	2e-5	4-5	Freeze bottom 8
Generation (GPT)			
Completion	2e-5	2-3	Full fine-tuning
Dialogue	1e-5	3-5	Full fine-tuning
Summarization	3e-5	2-3	Full fine-tuning

Batch size: 16-32, Warmup: 10% of steps

Start with these proven recipes, then experiment

Appendix E: Training Cost Analysis

Pre-training Costs (2018-2024):

Model	Cost
BERT-base	$7K
BERT-large	$25K
GPT-1	$50K
GPT-2	$500K
GPT-3	$4.6M
GPT-4 (est)	$50M+

Why So Expensive:

- Massive datasets (100B-1T tokens)
- Large models (1B-1T parameters)
- Weeks/months on thousands of GPUs
- Trial and error in hyperparameters

Fine-tuning Costs (Per Task):

Dataset Size	Cost
100 examples	$5-10
1,000 examples	$50-100
10,000 examples	$200-500

The Economics:

- Pre-training: One-time investment
- Fine-tuning: Cheap per task
- Amortize cost across many applications

Business Model:
OpenAI, Anthropic, Google: Pay for pre-training, sell API access

Pre-training economics enable the modern AI industry

Appendix F: Further Reading and Resources

Original Papers:

- **Attention**: Vaswani et al. (2017)
  "Attention is All You Need"
- **GPT-1**: Radford et al. (June 2018)
  "Improving Language Understanding by Generative Pre-Training"
- **BERT**: Devlin et al. (October 2018)
  "BERT: Pre-training of Deep Bidirectional Transformers"
- **GPT-2**: Radford et al. (2019)
  "Language Models are Unsupervised Multitask Learners"
- **GPT-3**: Brown et al. (2020)
  "Language Models are Few-Shot Learners"

Practical Resources:

- **HuggingFace Transformers**
  Library for using pre-trained models
  https://huggingface.co/transformers
- **Model Hub**
  Thousands of pre-trained models
  https://huggingface.co/models
- **Fine-tuning Tutorials**
  Official guides for common tasks
- **Papers With Code**
  Leaderboards and implementations

Next Week:
Advanced architectures (T5, GPT-4, etc.)

These papers are essential reading for serious NLP practitioners