Natural Language Processing Course

Week 6: Pre-trained Language Models

Joerg R. Osterrieder
www.joergosterrieder.com

Week 6

Pre-trained Language Models

Learning from All of Human Knowledge

The $10 Million Problem No One Talked About

Training a language model from scratch (2018): Company A trains model for sentiment analysis:

- Cost: $500,000 in compute[1]
- Time: 2 weeks on 64 GPUs
- Learns: Grammar, syntax, word meanings, sentiment

Company B trains model for question answering:

- Cost: $500,000 in compute
- Time: 2 weeks on 64 GPUs
- Learns: Grammar, syntax, word meanings... again!

Every team re-learning what "the" means from scratch!

The waste: 90% of training teaches same basic language understanding

[1] Based on 2018 cloud GPU pricing and typical training times

The Revolution: Learn Language Once, Use Everywhere

The breakthrough idea (2018):[2]

What if we:

1. Train ONE model on massive text (Wikipedia, books, web)
2. Learn general language understanding
3. Share this pre-trained model with everyone
4. Fine-tune for specific tasks with little data

The impact:

- Training cost: \$10M → \$100[3]
- Training time: Weeks → Hours
- Data needed: 1M examples → 1K examples
- Performance: 70% → 95% accuracy

Transfer Learning: Don't start from scratch, start from knowledge!

[1] Howard & Ruder (2018) ULMFiT; Radford et al. (2018) GPT; Devlin et al. (2019) BERT

[2] Fine-tuning costs vs pre-training from scratch

Pre-trained Models Power Everything (2024)

Search & Understanding:

- Google Search: BERT since 2019[4]
- Affects 10% of all queries
- Bing: Multiple BERT variants
- Every modern search engine

Writing Assistants:

- Grammarly: BERT-based
- Google Docs: Smart compose
- Microsoft Editor
- All use pre-trained models

Business Applications:

- Customer service: 80% automated[5]
- Document analysis
- Email classification
- Resume screening

Key Models:

- BERT: 110M-340M parameters
- GPT-2: 1.5B parameters
- RoBERTa: Better BERT training
- T5: Unified text-to-text

2024: You never train from scratch - always start from pre-trained

[1] Google blog: "Understanding searches better than ever before"
[2] Gartner report on AI automation

Week 6: What You'll Master

By the end of this week, you will:

- **Understand** why pre-training changes everything
- **Master** BERT's bidirectional approach
- **Implement** masked language modeling
- **Compare** BERT vs GPT architectures
- **Fine-tune** models for your own tasks

Core Insight: Language understanding is a reusable skill

Pre-training: Learning from Raw Text

Traditional supervised learning:

- Need: Labeled data (expensive!)
- Example: 100K sentiment labels
- Problem: Starts from random weights

Self-supervised pre-training:

- Need: Just text (free and abundant!)
- Example: All of Wikipedia (6B tokens)
- Advantage: Learns language before task

The clever trick - Create labels from text itself:

1. Take: "The cat sat on the [?]"
2. Model predicts: "mat"
3. No human labeling needed!
4. Can use internet-scale data

Pre-training = Teaching models to read before teaching them tasks

BERT: Bidirectional Understanding

The limitation of GPT (left-to-right):
"The man went to the [MASK] to buy milk"

- GPT only sees: "The man went to the"
- Can't use "to buy milk" as context!

BERT's innovation: Look both ways![6]

- Sees entire sentence: "The man went to the [MASK] to buy milk"
- Can use both left AND right context
- Predicts: "store" (using "buy milk" as clue)

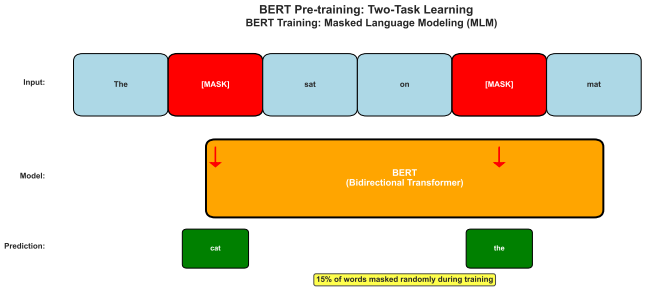
Masked Language Model (MLM):

1. Randomly mask 15% of tokens
2. Predict masked tokens using all context
3. Learn deep bidirectional representations

BERT = Transformer encoder + Bidirectional context

[6] Devlin et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers"

BERT's Training Process Visualized



GPT: Generative Pre-training

Different philosophy: Predict the future[7]

Given: "The cat sat on the"
Predict: "mat" (then "in", then "the", then "sun"...)

Autoregressive training:

1. Process text left-to-right
2. Predict next token at each step
3. Can generate coherent text!
4. Same objective as traditional LM

Key differences from BERT:

- Unidirectional (left-to-right only)
- Can generate text naturally
- Simpler training (no masking tricks)
- Transformer decoder architecture

GPT = Transformer decoder + Autoregressive training

[7] Radford et al. (2018). "Improving Language Understanding by Generative Pre-Training"

Implementing BERT's Masked Language Model

```python
import torch
import torch.nn as nn
import random

class BERTMasking:
    def __init__(self, vocab_size, mask_token_id, mask_prob=0.15):
        """BERT-style masking for MLM"""
        self.vocab_size = vocab_size
        self.mask_token_id = mask_token_id
        self.mask_prob = mask_prob

    def mask_tokens(self, inputs, special_tokens_mask=None):
        """Mask tokens for BERT training"""
        labels = inputs.clone()

        # Create probability matrix
        probability_matrix = torch.full(labels.shape, self.mask_prob
            )
        if special_tokens_mask is not None:
            probability_matrix.masked_fill_(special_tokens_mask,
                value=0.0)

        masked_indices = torch.bernoulli(probability_matrix).bool()
        labels[~masked_indices] = -100  # Only compute loss on
            masked

        # 80% of time, replace with [MASK]
        indices_replaced = torch.bernoulli(
            torch.full(labels.shape, 0.8)).bool() & masked_indices
        inputs[indices_replaced] = self.mask_token_id

        # 10% of time, replace with random word
        indices_random = torch.bernoulli(
            torch.full(labels.shape, 0.5)).bool() & masked_indices &
                ~indices_replaced
```

BERT's 80-10-10 Rule:

- 80%: Replace with [MASK]
- 10%: Replace with random token
- 10%: Keep original token

Why this complexity?

MASK never seen in fine-tuning

- Random replacement adds noise
- Original tokens prevent mismatch

Design validates through:

- Ablation studies in paper
- 15% masking is optimal
- Too much masking hurts learning

BERT Model Architecture

```python
class BERTModel(nn.Module):
    def __init__(self, vocab_size, hidden_size=768, num_layers=12,
                 num_heads=12, max_length=512, dropout=0.1):
        """BERT model for masked language modeling"""
        super().__init__()

        # Token embeddings + position + segment
        self.token_embeddings = nn.Embedding(vocab_size, hidden_size)
        self.position_embeddings = nn.Embedding(max_length, hidden_size)
        self.segment_embeddings = nn.Embedding(2, hidden_size)

        self.layer_norm = nn.LayerNorm(hidden_size)
        self.dropout = nn.Dropout(dropout)

        # Transformer encoder layers
        encoder_layer = nn.TransformerEncoderLayer(
            hidden_size, num_heads, dim_feedforward=4*hidden_size,
            dropout=dropout, activation='gelu'
        )
        self.transformer = nn.TransformerEncoder(encoder_layer, num_layers)

        # MLM head
        self.mlm_head = nn.Sequential(
            nn.Linear(hidden_size, hidden_size),
            nn.GELU(),
            nn.LayerNorm(hidden_size),
            nn.Linear(hidden_size, vocab_size)
        )

    def forward(self, input_ids, segment_ids=None, attention_mask=None):
        """Forward pass for MLM"""
```

BERT Sizes:

- BASE: 12 layers, 768 hidden, 110M params[8]
- LARGE: 24 layers, 1024 hidden, 340M params

Key Components:

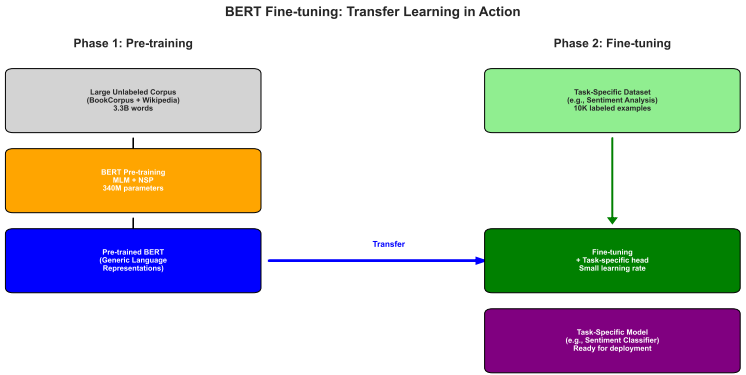
- Segment embeddings for sentence pairs
- GELU activation (smoother than ReLU)
- Layer norm before attention

Most applications use BERT-BASE

Fine-tuning: From General to Specific

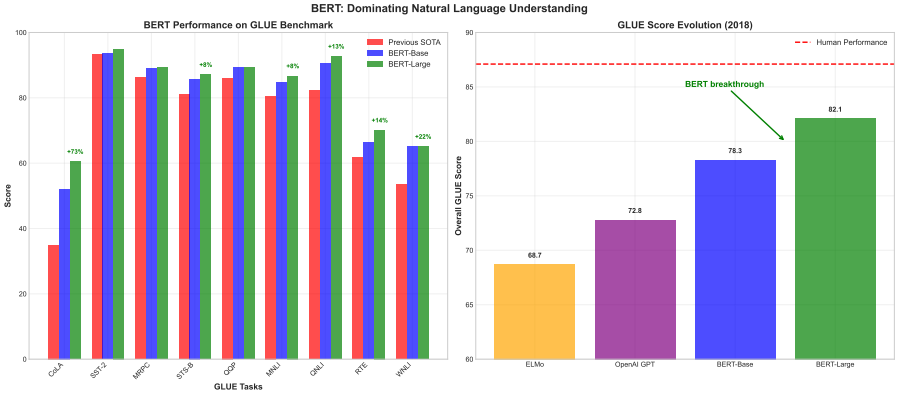
The fine-tuning recipe:

1. Start with pre-trained BERT/GPT
2. Add task-specific head (1-2 layers)
3. Train on your labeled data
4. 100x less data needed!



Comparison: From Scratch vs Fine-tuning

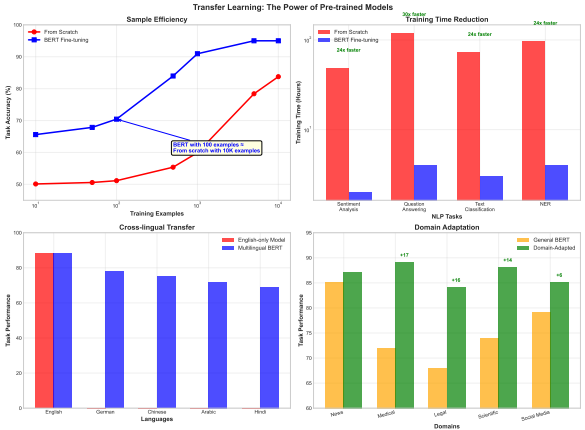
Pre-training Impact: BERT Dominates Every Task



Key Insights

- GLUE benchmark: 9 language understanding tasks
- BERT improved SOTA on ALL tasks simultaneously
- Average score: 82.1 (previous best: 75.1)
- Some tasks: 10+ point improvements

The Power of Transfer Learning



Key insights:

- With 100 examples: BERT matches 10K from-scratch training
- Especially powerful for low-resource languages
- Works across very different tasks
- The more diverse pre-training, the better transfer

The Pre-trained Model Zoo (2024)

General Purpose:

- BERT: Understanding tasks
- GPT-2/3: Generation tasks
- T5: Any text-to-text task
- ELECTRA: Efficient BERT alternative

Domain Specific:

- BioBERT: Biomedical text
- SciBERT: Scientific papers
- FinBERT: Financial documents
- CodeBERT: Programming code

Multilingual:

- mBERT: 104 languages
- XLM-R: 100 languages
- mT5: Text-to-text multilingual

Where to find:

- Hugging Face: 500K+ models[9]
- Google TF Hub
- PyTorch Hub
- Company model hubs

2024 Reality: Never train from scratch - there's a model for that!

[9] Hugging Face hosts majority of open models

Pre-trained Model Pitfalls

1. Domain Mismatch

- Problem: BERT trained on books/wiki, you have tweets
- Solution: Domain-adaptive pre-training
- Continue pre-training on your domain

2. Catastrophic Forgetting

- Problem: Fine-tuning destroys pre-trained knowledge
- Solution: Lower learning rates (2e-5), gradual unfreezing

3. Overfitting on Small Data

- Problem: Large model, small dataset
- Solution: Freeze lower layers, only train top layers

Real Example - Customer Service Bot:

- Started with BERT: 72% accuracy
- Continued pre-training on support tickets: 85%
- Fine-tuned with careful LR: 94%

Week 6 Exercise: Fine-tune Your Own BERT

Your Mission: Take pre-trained BERT and make it expert at your task

Part 1: Understand Pre-training

- Load pre-trained BERT from Hugging Face
- Examine what it already knows (probe tasks)
- Visualize attention patterns

Part 2: Fine-tune for Classification

- Choose: Sentiment, spam, or topic classification
- Add classification head to BERT
- Compare: 100 vs 1000 vs 10000 examples
- Track how quickly it learns

Part 3: Advanced Experiments

- Try different learning rates
- Freeze vs unfreeze layers
- Compare BERT vs training from scratch
- Measure training time savings

You'll discover: Why no one trains from scratch anymore!

Key Takeaways: The Pre-training Revolution

What we learned:

- Training from scratch wastes resources
- Pre-training learns reusable language understanding
- BERT: Bidirectional with masking
- GPT: Autoregressive generation
- Fine-tuning needs 100x less data

The paradigm shift:

Task-specific training $\rightarrow$ Pre-train then fine-tune

Why it matters:

- Democratized NLP (anyone can use BERT)
- Enabled rapid prototyping
- Made NLP practical for businesses

Next week: Advanced Transformers
How do we scale to GPT-3's 175B parameters and beyond?

References and Further Reading

Foundational Papers:

- Devlin et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers"
- Radford et al. (2018). "Improving Language Understanding by Generative Pre-Training"
- Liu et al. (2019). "RoBERTa: A Robustly Optimized BERT"

Practical Resources:

- Hugging Face Transformers library
- "The Illustrated BERT" by Jay Alammar
- Google's BERT GitHub repository

Recent Advances:

- ELECTRA: More efficient pre-training
- DeBERTa: Disentangled attention
- Transfer learning survey (Qiu et al., 2020)