# Lecture 18: BERT Deep Dive

## Week 6, Lecture 1 - Bidirectional Encoder Representations

### PSYC 51.07: Models of Language and Communication

Winter 2026

# Today's Agenda 📋

1. 🎯 **BERT Introduction**: What makes it special?
2. 🎭 **Masked Language Modeling**: The key training objective
3. 🏗️ **BERT Architecture**: Model sizes and specifications
4. 📊 **Pre-training & Fine-tuning**: The two-stage paradigm
5. 🔬 **Contextual Embeddings**: Seeing polysemy in action
6. 💻 **Using BERT**: Practical code examples

*Goal: Deep understanding of BERT and how it revolutionized NLP*

# BERT: Bidirectional Encoder Representations 🎯

**BERT = Encoder-only Transformer**

**Key Innovation: Masked Language Modeling (MLM)**

**Traditional Language Models:**

- Left-to-right (GPT)
- Or right-to-left
- Can't see full picture

**Example:**

- "The cat sat on the ___"
- Only sees left context



**BERT (MLM):**

# Why BERT Was Revolutionary 🚀

**Before BERT (2018):**

- Feature-based approaches (use Word2Vec/GloVe as features)
- Task-specific architectures
- Limited transfer learning
- Unidirectional or shallow bidirectional models

**BERT's Contributions:**

1. **Deep Bidirectionality**
    - True bidirectional context at every layer

- Not just concatenating left-to-right and right-to-left

2. **Pre-train + Fine-tune Paradigm**

# Masked Language Modeling (MLM) 🎭

## BERT's Pre-training Objective

**Training Procedure:**

1. Take a sentence
2. Randomly mask 15% of tokens
3. Of the masked tokens:
    - 80%: Replace with [MASK]

- 10%: Replace with random word
- 10%: Keep unchanged

4. Predict the original tokens

### Example

# MLM Example: Step by Step 📝

**Sentence:** "The quick brown fox jumps over the lazy dog"

### Step 1: Select Tokens (15%)

9 tokens total, mask ~1-2

```
tokens = ["The", "quick", "brown", "fox",
          "jumps", "over", "the", "lazy", "dog"]
# Randomly select: "quick" (idx 1), "over" (idx 5)
```

### Step 2: Apply 80/10/10 Strategy

```
"quick" → 80% → [MASK]
"over"  → 10% → "under" (random)
```



### Step 3: Create Training Example

# Next Sentence Prediction (NSP) 🔗

**BERT's second pre-training objective (debated usefulness)**

**Task:** Given two sentences A and B, predict if B follows A

Positive Example (IsNext)

**Sentence A:** "The man went to the store."

**Sentence B:** "He bought a gallon of milk."

**Label:** IsNext ✓

Negative Example (NotNext)

**Sentence A:** "The man went to the store."

**Sentence B:** "Penguins are flightless birds."

**Label:** NotNext ✕

# BERT Architecture Variants 📊

| | | | | |
|---|---|---|---|---|
| **BERT-Large** | **24** | **1024** | **16** | **340M** |

**Architecture Details (BERT-Base):**

- 12 transformer encoder layers
- 768-dimensional hidden states
- 12 attention heads per layer (64 dims each)
- 3072-dimensional feed-forward intermediate size (4x expansion)
- Maximum sequence length: 512 tokens
- Vocabulary size: 30,000 WordPiece tokens

**Special Tokens:**

- **[CLS]**: Classification token (first token, used for sequence-level tasks)

# BERT Input Representation

**Three types of embeddings are summed:**

```
# Example: Sentence pair for NSP
sentence_a = "My dog is cute"
sentence_b = "He likes playing"

# Tokenization
tokens = ["[CLS]", "my", "dog", "is", "cute", "[SEP]", "he", "likes", "playing"

# Three embedding types (each is a 768-dim vector):
token_emb   = [E_CLS, E_my, E_dog, E_is, E_cute, E_SEP, E_he, E_likes, E_playin
segment_emb = [E_A,   E_A,  E_A,   E_A,  E_A,    E_A,   E_B,  E_B,     E_B,
position_emb= [E_0,   E_1,  E_2,   E_3,  E_4,    E_5,   E_6,  E_7,     E_8,

# Final input = token + segment + position (element-wise sum)
input_embedding = token_emb + segment_emb + position_emb
```





**Three embedding types:**

# WordPiece Tokenization: Worked Example 🔤

## How BERT handles unknown words

```
from transformers import BertTokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Common words stay intact
tokenizer.tokenize("The cat sat on the mat")
# → ['the', 'cat', 'sat', 'on', 'the', 'mat']

# Rare/unknown words get split into subwords
tokenizer.tokenize("unbelievably")
# → ['un', '##believable', '##ly']  # "##" means continuation

tokenizer.tokenize("ChatGPT is transformative")
# → ['chat', '##g', '##pt', 'is', 'transform', '##ative']
```

## Why WordPiece?

# BERT Pre-training 🏋️

**Massive scale pre-training on unlabeled text**

**Pre-training Data:**

- **BooksCorpus**: 800M words (novels, fiction)
- **English Wikipedia**: 2,500M words
- Total: 3.3 billion words
- Diverse, high-quality text

**Training Details:**

- Batch size: 256 sequences (128,000 tokens)
- Training steps: 1M steps
- Optimization: Adam (lr=1e-4, warmup=10k steps)

# Fine-tuning BERT 🎓

### Two-stage process: Pre-train then Fine-tune

**Stage 1: Pre-training** (done once)

```
# Expensive: weeks on TPUs
# Data: 3.3B words (books + Wikipedia)
# Task: MLM + NSP
# Result: General language understanding

model = pretrain_bert(
    data=["BooksCorpus", "Wikipedia"],
    steps=1_000_000,
    hardware="16 TPUs"
)
```

**Stage 2: Fine-tuning** (per task)

# Fine-tuning for Different Tasks 🎯

**Minimal architecture changes needed!**

1. **Single Sentence Classification**
   - Input: [CLS] sentence [SEP]

- Output: [CLS] representation → classifier
- Example: Sentiment analysis

2. **Sentence Pair Classification**
   - Input: [CLS] sentence A [SEP] sentence B [SEP]

- Output: [CLS] representation → classifier
- Example: Natural Language Inference

3. **Question Answering**
   - Input: [CLS] question [SEP] passage [SEP]

# Fine-tuning BERT: Code Example 💻

## Using HuggingFace Transformers

```
from transformers import BertForSequenceClassification, Trainer, TrainingArgume

# Load pre-trained BERT with classification head
model = BertForSequenceClassification.from_pretrained(
    'bert-base-uncased',
    num_labels=2  # Binary classification
)

# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=16,
    learning_rate=2e-5,
    warmup_steps=500,
)
```

# Fine-tuning BERT: Code Example 💻

```
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
)

trainer.train()
```

...continued

*Reference: HuggingFace Course - Chapters 1.5, 7.3*

# Contextual Embeddings in Action 🔬

**Remember "bank"? Let's see BERT handle it!**

```
from transformers import BertTokenizer, BertModel
import torch

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# Two different contexts for "bank"
sent1 = "I deposited money at the bank"
sent2 = "We sat by the river bank"

# Get embeddings
def get_embedding(sentence, target_word):
    inputs = tokenizer(sentence, return_tensors='pt')
    outputs = model(**inputs)
    # Find position of target word
    tokens = tokenizer.tokenize(sentence)
```

# Contextual Embeddings in Action 🔬

```
emb2 = get_embedding(sent2, "bank")  # River bank

# Compare similarity
similarity = torch.cosine_similarity(emb1, emb2, dim=0)
print(f"Similarity: {similarity:.3f}")  # Low! (~0.3-0.5)
# Different contexts → Different embeddings!
```

...continued

# Visualizing BERT's Contextual Embeddings 📊

## Same word, different meanings, different vectors

```python
# Find nearest neighbors for "bank" in each context
from sklearn.neighbors import NearestNeighbors

# Financial "bank" context
neighbors_financial = find_nearest_words(emb1, vocabulary)
# → ["banks", "financial", "account", "deposit", "loan", "credit"]

# River "bank" context
neighbors_river = find_nearest_words(emb2, vocabulary)
# → ["shore", "riverside", "banks", "stream", "water", "edge"]
```

### Concrete Measurements

| Word Pair | Word2Vec Similarity | BERT Similarity |
|---|---|---|
| bank (fin) vs bank (river) | 1.00 (same vector!) | 0.42 |

# BERT's Impressive Results 📈

**State-of-the-art on 11 NLP tasks when released (2018)**

| SQuAD 2.0 (QA) | F1 | 66.3 | 83.1 |
|---|---|---|---|
| MNLI (NLI) | Accuracy | 80.6 | 86.7 |
| SST-2 (Sentiment) | Accuracy | 93.2 | 94.9 |
| CoNLL-2003 (NER) | F1 | 92.6 | 92.8 |

**Key Observations:**

- Largest gains on tasks requiring understanding (QA, NLI)
- Improvements even on well-studied benchmarks
- BERT-Large generally better than BERT-Base
- Fine-tuning is simple but very effective

# What Does BERT Learn? 🧠

**Probing BERT's internal representations**

**Research has shown BERT captures:**

1. **Syntactic Information**
    - Part-of-speech tags

- Constituent structure
- Dependency relations
- Lower layers encode more syntax

2. **Semantic Information**
    - Word sense disambiguation

- Semantic roles

# BERT Layer Analysis 📊

## Different layers capture different linguistic properties

```
# Probing experiment: Train linear classifiers on each layer's representations
from transformers import BertModel
import numpy as np

model = BertModel.from_pretrained('bert-base-uncased', output_hidden_states=Tru

# Get hidden states for all 12 layers
outputs = model(**inputs)
hidden_states = outputs.hidden_states  # (13 layers: embedding + 12 transformer

# Results from probing studies (Tenney et al., 2019):
layer_specialization = {
    "Layers 0-2":  ["POS tagging", "Word boundaries"],      # Surface
    "Layers 3-6":  ["Parse trees", "Dependencies"],         # Syntax
    "Layers 7-9":  ["Semantic roles", "Coreference"],       # Semantics
    "Layers 10-12": ["Task-specific representations"]        # Task
```

# Discussion Questions

1. **MLM vs. Autoregressive:**
    - Why is MLM better for understanding tasks?

- Can BERT generate text like GPT?
- What are the trade-offs?

2. **The 80/10/10 Masking Strategy:**
    - Why not just use 100% [MASK]?

- What problem does the random replacement solve?
- Could we improve this strategy?

3. **Pre-training Data:**
    - Why use books and Wikipedia?

- Would social media text work as well?

# Looking Ahead 🔮

**Today we learned:**

- BERT architecture and innovations
- Masked Language Modeling
- Pre-training and fine-tuning
- Contextual embeddings
- What BERT learns

**Next lecture (Lecture 16 - BERT Variants):**

- : Optimized BERT training
- : Parameter-efficient BERT
- : Smaller, faster BERT
- : ELECTRA, DeBERTa, and more

# Summary 🎯

**Key Takeaways:**

1. **BERT = Encoder-only Transformer**
    - Bidirectional self-attention

- Trained with Masked Language Modeling

2. **Pre-train + Fine-tune Paradigm**
    - Expensive pre-training on unlabeled data (once)

- Cheap fine-tuning on task-specific data (per task)

3. **Contextual Embeddings**
    - Different representations based on context

- Solves polysemy problem

# References 📚

**Essential Papers:**

- **Devlin et al. (2019)** - "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"
- The original BERT paper
- Introduced MLM and NSP

  \item **Tenney et al. (2019)** - "BERT Rediscovers the Classical NLP Pipeline"

  - Analysis of what BERT learns
- Layer-wise linguistic properties

  \item **Clark et al. (2019)** - "What Does BERT Look At? An Analysis of BERT's Attention"

# Questions? 🙋

**Discussion Time**

**Topics for discussion:**

- Masked Language Modeling
- Pre-training vs fine-tuning
- Contextual embeddings
- BERT architecture details
- Implementation questions

Thank you! 🙏

Next: BERT Variants and Improvements!