Contextual Embeddings: ELMo, USE, BERT

Lecture 12: Beyond Static Word Representations

PSYC 51.07: Models of Language and Communication - Week 4

Winter 2026

Today's Lecture 📋

1. 🔄 **From Static to Contextual**
2. 🧬 **Language Models as Feature Extractors**
3. 🔮 **ELMo: Embeddings from Language Models**
4. 🌐 **Universal Sentence Encoder**
5. 🎭 **BERT: Bidirectional Transformers**
6. 📊 **Comparison & Applications**

Goal: Understand how context transforms word representation

The Polysemy Problem Revisited 🤔

Recall: Static embeddings assign ONE vector per word

Example: "bank"

1. "I deposited money at the **bank**"
   (financial institution)
2. "We sat by the river **bank**"
   (riverside)
3. "The plane will **bank** left"
   (tilt/turn)

Word2Vec/GloVe: All three get the SAME vector!

Contextual embeddings: Each gets a DIFFERENT vector based on context

Cosine Similarity Demo:

Static vs. Contextual Embeddings 🔄

Static (Word2Vec, GloVe, FastText):

```
# Same vector every time
model = Word2Vec(...)
vec1 = model['bank']
vec2 = model['bank']

assert vec1 == vec2  # True!
```

Characteristics:

- One vector per word type
- Context-independent
- Fast lookup (dictionary)
- Fixed after training

Language Models as Feature Extractors 🧬

Key Insight: Train a language model, use its internal states as embeddings

Language Modeling Task:

Predict the next word given previous words: $P(w_t | w_1, w_2, \ldots, w_{t-1})$

Worked Example

Input: "The cat sat on the ___"

Model predicts probabilities:	
"mat"	0.25
"floor"	0.18
"couch"	0.12
"dog"	0.001

ELMo: Embeddings from Language Models 🔮

The first widely-adopted contextual embedding (2018)

Key Ideas:

1. Train deep bidirectional language model
2. Use all layer activations
3. Weighted combination per task
4. Pre-train on large corpus

Architecture:

- 2-layer biLSTM
- Forward LM: $P(w_t | w_1 \ldots w_{t-1})$
- Backward LM: $P(w_t | w_{t+1} \ldots w_n)$

ELMo: How It Works 🔍

Training:

1. Pre-train on large corpus (1B Word Benchmark)
2. Each position gets representation from all layers

Usage (downstream tasks):

1. Freeze ELMo weights
2. For each token, extract representations from all layers
3. Learn task-specific weighted combination
4. Concatenate with task model

Concrete Example: Sentiment Analysis

Input: "The movie was absolutely terrible"

ELMo in Practice 💻

```
from allennlp.modules.elmo import Elmo, batch_to_ids

# Initialize ELMo
options_file = "https://s3-us-west-2.amazonaws.com/allennlp/models/elmo/2x4096_
weight_file = "https://s3-us-west-2.amazonaws.com/allennlp/models/elmo/2x4096_5

elmo = Elmo(options_file, weight_file, 2, dropout=0)

# Prepare sentences
sentences = [
    ['I', 'deposited', 'money', 'at', 'the', 'bank'],
    ['We', 'sat', 'by', 'the', 'river', 'bank']
]

# Convert to character ids
character_ids = batch_to_ids(sentences)

# Get embeddings
```

ELMo in Practice 💻

```
# embeddings['elmo_representations'] contains:
# - List of 2 tensors (one per layer)
# - Shape: [batch_size, seq_len, 1024]

# Different vectors for "bank"!
bank1 = embeddings['elmo_representations'][0][0, 5, :]  # first sentence
bank2 = embeddings['elmo_representations'][0][1, 5, :]  # second sentence

# Cosine similarity will be lower than for static embeddings
```

...continued

Universal Sentence Encoder (USE) 🌐

Sentence-level embeddings for semantic similarity

Motivation:

- Word embeddings: good for words
- But what about sentences?
- Average of word vectors? Too simple!
- Need compositionality

Two Variants:

1. Transformer-based:

- Higher accuracy
- Slower (compute intensive)

Universal Sentence Encoder in Practice 💻

```python
import tensorflow_hub as hub
import numpy as np

# Load model
embed = hub.load("https://tfhub.dev/google/universal-sentence-encoder/4")

# Example sentences
sentences = [
    "The cat sat on the mat.",
    "A feline rested on the rug.",
    "The dog ran in the park.",
    "I love machine learning."
]

# Generate embeddings
embeddings = embed(sentences)

# Shape: [4, 512]
```

Universal Sentence Encoder in Practice 💻

```python
# Compute similarity
from sklearn.metrics.pairwise import cosine_similarity

sim_matrix = cosine_similarity(embeddings)
print(sim_matrix)

# Sentences 1 and 2 should be very similar (paraphrases)
# Sentence 3 somewhat similar (animals)
# Sentence 4 dissimilar

# Use for semantic search
query = "cat on mat"
query_embedding = embed([query])
similarities = cosine_similarity(query_embedding, embeddings)[0]
most_similar_idx = np.argmax(similarities)
print(f"Most similar: {sentences[most_similar_idx]}")
```


continued

BERT: Bidirectional Encoder Representations 🎭

The model that changed everything (2018)

Key Innovations:

1. context (not just left-to-right)
2. architecture (attention)
3. pre-training
4. Deeply bidirectional

Impact:

- SOTA on 11 NLP tasks
- Sparked the "BERT-era"
- 1000+ variants (RoBERTa, ALBERT, DistilBERT, ...)

Masked Language Modeling (MLM) 🎭

BERT's key training innovation

The Problem with Traditional LM:

- Left-to-right: Only sees previous words
- Right-to-left: Only sees next words
- Want: See both directions simultaneously
- But: Can't just show the answer during training!

Solution: Mask some words, predict them

Worked Example: MLM Training

Original: "The cat sat on the mat"

Step 1: Randomly select 15% of tokens → "cat" selected

Next Sentence Prediction (NSP) 🔗

Second pre-training task: Understand sentence relationships

Task: Given two sentences A and B, predict if B follows A in the text

Positive Example (IsNext)

Sentence A: "The cat sat on the mat."

Sentence B: "It was sleeping peacefully."

Label: IsNext ✓

Negative Example (NotNext)

Sentence A: "The cat sat on the mat."

Sentence B: "Machine learning is fascinating."

Label: NotNext ✕

BERT Architecture 🏗️

Three types of embeddings are summed for each token:

Worked Example: Input Representation

Input: "[CLS] I love NLP [SEP] It is fun [SEP]"

Token	Token ID	Segment	Position	Final Embedding
[CLS]	E_CLS	A	0	E_CLS + E_A + E_0
I	E_I	A	1	E_I + E_A + E_1
love	E_love	A	2	E_love + E_A + E_2
NLP	E_NLP	A	3	E_NLP + E_A + E_3
[SEP]	E_SEP	A	4	E_SEP + E_A + E_4
It	E_It	B	5	E_It + E_B + E_5

BERT Architecture 🏗️

Token	Token ID	Segment	Position	Final Embedding
[SEP]	E_SEP	B	8	E_SEP + E_B + E_8

...continued

- **Token Embedding:** What word is this?
- **Segment Embedding:** Which sentence (A or B)?
- **Position Embedding:** Where in the sequence?

BERT in Practice 💻

```python
from transformers import BertTokenizer, BertModel
import torch

# Load pre-trained BERT
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

# Example sentences with "bank"
sent1 = "I deposited money at the bank"
sent2 = "We sat by the river bank"

# Tokenize
tokens1 = tokenizer(sent1, return_tensors='pt')
tokens2 = tokenizer(sent2, return_tensors='pt')

# Get embeddings
with torch.no_grad():
    output1 = model(**tokens1)
```

BERT in Practice 💻

```python
# Last hidden state: [batch_size, seq_len, hidden_size]
embeddings1 = output1.last_hidden_state
embeddings2 = output2.last_hidden_state

# Extract "bank" embedding (position varies)
# tokens1: [CLS] i deposited money at the bank [SEP]
bank1_embedding = embeddings1[0, 6, :]  # 768-dim vector

# tokens2: [CLS] we sat by the river bank [SEP]
bank2_embedding = embeddings2[0, 6, :]  # 768-dim vector

# Different vectors for "bank"!
from torch.nn.functional import cosine_similarity
sim = cosine_similarity(bank1_embedding, bank2_embedding, dim=0)
print(f"Similarity: {sim:.3f}")  # Lower than with static embeddings
```

...continued

Fine-tuning BERT 🎯

Two ways to use BERT:

1. Feature Extraction:

- Freeze BERT weights
- Use embeddings as features
- Train classifier on top
- Faster, less data needed

```
# Freeze BERT
for param in bert_model.parameters():
    param.requires_grad = False

# Add classifier
classifier = nn.Linear(768, num_classes)
```

Contextual Embeddings Comparison 📊

Pre-training	LM (forward+backward)	Multi-task	MLM + NSP
Bidirectional	Shallow	Yes	Deep
Granularity	Token	Sentence	Token
Hidden size	1024	512	768/1024
Parameters	93M	256M	110M/340M
Speed	Medium	Fast	Slow
OOV handling	Characters	Subwords	WordPiece
Year	2018	2018	2018

Recommendations

- **ELMo:** Legacy systems, character-aware needs

Impact on NLP 🌟

BERT revolutionized NLP:

Before BERT (pre-2018):

- Task-specific architectures
- Train from scratch
- Static embeddings (Word2Vec, GloVe)
- Limited transfer learning
- Moderate performance

Tasks that improved:

- Question Answering (+1.5 F1 on SQuAD)
- NER (+0.3 F1)

Real-World Applications 🚀

1. Google Search:

```
Query: "can you get medicine for
         someone pharmacy"

BERT understands: picking up a
prescription FOR someone else

Before BERT: matched "medicine"
and "pharmacy" keywords only
```

2. Question Answering:

```
Context: "The Eiffel Tower was
built in 1889 by Gustave Eiffel."

```

Discussion Question 💬

Do contextual embeddings truly "understand" language?

Consider:

- BERT can distinguish "bank" (financial) from "bank" (riverside)
- It achieves human-level performance on many benchmarks
- But it's trained only on text co-occurrence patterns

Arguments For:

- Captures complex semantic relationships
- Generalizes to new contexts
- Emergent linguistic capabilities
- Handles compositional meaning

Practical Tips 💡

1. **Choosing a Model:**
    - BERT-base: Good balance, 110M params

- DistilBERT: 40% smaller, 60% faster, 97% performance
- RoBERTa: Better than BERT, longer training
- Domain-specific: BioBERT, SciBERT, FinBERT, etc.

2. **Fine-tuning Best Practices:**
    - Small learning rate (2e-5 typical)

- Few epochs (2-4)
- Batch size: 16 or 32
- Warm-up steps
- Gradient clipping

Summary 🎯

What we learned today:

1. **Contextual vs. Static:** Different vectors per occurrence
2. **ELMo (2018):**
    - BiLSTM language models

- Character-based, handles OOV
- Task-specific weighting

3. **Universal Sentence Encoder (2018):**
    - Sentence-level embeddings

- Two variants: Transformer & DAN
- Optimized for semantic similarity

4. **BERT (2018)**

Key References 📚

Foundational Papers:

- Peters et al. (2018). "Deep contextualized word representations" (ELMo)
- Cer et al. (2018). "Universal Sentence Encoder"
- Devlin et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"

BERT Variants:

- Liu et al. (2019). "RoBERTa: A Robustly Optimized BERT Pretraining Approach"
- Sanh et al. (2019). "DistilBERT, a distilled version of BERT"
- Lan et al. (2019). "ALBERT: A Lite BERT for Self-supervised Learning"

Critical Perspectives:

Questions? 🙋

Next Lecture:

Dimensionality Reduction: PCA, t-SNE, UMAP

Visualizing high-dimensional embeddings!