CSC 4315: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Week 12, Lecture 23

Text Processing, Embeddings, and Word2Vec

Lecture Overview

This lecture introduces natural language processing fundamentals including tokenization, word embeddings, and the Word2Vec algorithm.

Topics Covered:

- Text preprocessing pipeline
- One-hot vs dense embeddings
- Word2Vec: Skip-gram and CBOW
- Pre-trained embeddings
- Simple text classification

1. Text Preprocessing

1.1 Text to Numbers

```
import torch
import torch.nn as nn
import numpy as np

print("Text Preprocessing")
print("="*70)

print("Challenge: Neural networks need numbers, not text!")

print("\nPreprocessing Pipeline:")
print("  1. Tokenization: Split text into tokens")
print("  2. Vocabulary: Map tokens to integers")
print("  3. Embedding: Map integers to dense vectors")

# Simple tokenization
text = "The cat sat on the mat."
tokens = text.lower().replace(".", "").split()
print(f"\nText: '{text}'")
print(f"Tokens: {tokens}")

# Build vocabulary
vocab = {"&lt;PAD&gt;": 0, "&lt;UNK&gt;": 1}
for token in tokens:
    if token not in vocab:
        vocab[token] = len(vocab)

print(f"\nVocabulary: {vocab}")

# Convert to indices
indices = [vocab.get(t, vocab["&lt;UNK&gt;"]) for t in tokens]
print(f"Indices: {indices}")

print("\nTokenization approaches:")
print("  - Word-level: 'playing' -&gt; ['playing']")
print("  - Subword (BPE): 'playing' -&gt; ['play', 'ing']")
print("  - Character: 'cat' -&gt; ['c', 'a', 't']")
```

```
Text Preprocessing
======================================================================
Challenge: Neural networks need numbers, not text!

Preprocessing Pipeline:
  1. Tokenization: Split text into tokens
  2. Vocabulary: Map tokens to integers
  3. Embedding: Map integers to dense vectors

Text: 'The cat sat on the mat.'
Tokens: ['the', 'cat', 'sat', 'on', 'the', 'mat']

Vocabulary: {'&lt;PAD&gt;': 0, '&lt;UNK&gt;': 1, 'the': 2, 'cat': 3, 'sat': 4, 'on': 5, 'mat': 6}
Indices: [2, 3, 4, 5, 2, 6]

Tokenization approaches:
  - Word-level: 'playing' -&gt; ['playing']
  - Subword (BPE): 'playing' -&gt; ['play', 'ing']
  - Character: 'cat' -&gt; ['c', 'a', 't']
```

2. Word Embeddings

2.1 One-Hot vs Dense Embeddings

```
print("One-Hot vs Dense Embeddings")
print("="*70)

vocab_size = 10000

print("One-Hot Encoding:")
print(f"  Vector size: {vocab_size}")
print(f"  'cat' = [0, 0, ..., 1, ..., 0, 0]")
print("  Problems:")
print("    - Huge vectors (10,000+ dimensions)")
print("    - No similarity: cat · dog = 0")
print("    - Sparse and inefficient")

print("\nDense Embeddings:")
embed_dim = 300
print(f"  Vector size: {embed_dim}")
print(f"  'cat' = [0.2, -0.5, 0.8, ...]")
print("  Benefits:")
print("    - Compact representation")
print("    - Captures semantic similarity")
print("    - cat · dog &gt; 0 (both animals)")

# PyTorch embedding layer
embedding = nn.Embedding(vocab_size, embed_dim)
print(f"\nnn.Embedding({vocab_size}, {embed_dim})")
print(f"  Weight shape: {embedding.weight.shape}")
print(f"  Parameters: {embedding.weight.numel():,}")

# Lookup
indices = torch.tensor([42, 100, 5])
vectors = embedding(indices)
print(f"\nLookup indices {indices.tolist()}")
print(f"  Output shape: {vectors.shape}")
```

```
One-Hot vs Dense Embeddings
======================================================================
One-Hot Encoding:
  Vector size: 10000
  'cat' = [0, 0, ..., 1, ..., 0, 0]
  Problems:
    - Huge vectors (10,000+ dimensions)
    - No similarity: cat · dog = 0
    - Sparse and inefficient

Dense Embeddings:
  Vector size: 300
  'cat' = [0.2, -0.5, 0.8, ...]
  Benefits:
    - Compact representation
    - Captures semantic similarity
    - cat · dog &gt; 0 (both animals)

nn.Embedding(10000, 300)
  Weight shape: torch.Size([10000, 300])
  Parameters: 3,000,000

Lookup indices [42, 100, 5]
  Output shape: torch.Size([3, 300])
```

2.2 Word2Vec

```
print("Word2Vec (Mikolov et al., 2013)")
print("="*70)

print("Key idea: Words with similar contexts have similar meanings")
print("  'The cat sat on the ___' -&gt; mat, rug, floor")
print("  'The dog sat on the ___' -&gt; mat, rug, floor")
print("  =&gt; cat and dog have similar embeddings")
print("\nTwo architectures:")
print("\n1. Skip-gram: Predict context from center word")
print("   Input: 'cat' -&gt; Predict: 'the', 'sat', 'on'")
print("   Better for rare words")

print("\n2. CBOW (Continuous Bag of Words):")
print("   Input: 'the', 'sat', 'on' -&gt; Predict: 'cat'")
print("   Faster training")

print("\nTraining objective (Skip-gram):")
print("  maximize P(context | center)")
print("  = softmax(v_context · v_center)")

print("\nFamous property - Word Analogies:")
print("  king - man + woman ≈ queen")
print("  paris - france + italy ≈ rome")

print("\nUsing pre-trained embeddings:")
print("  - Word2Vec (Google)")
print("  - GloVe (Stanford)")
print("  - FastText (Facebook)")
```

```
Word2Vec (Mikolov et al., 2013)
======================================================================
Key idea: Words with similar contexts have similar meanings
  'The cat sat on the ___' -&gt; mat, rug, floor
  'The dog sat on the ___' -&gt; mat, rug, floor
  =&gt; cat and dog have similar embeddings

Two architectures:

1. Skip-gram: Predict context from center word
   Input: 'cat' -&gt; Predict: 'the', 'sat', 'on'
   Better for rare words

2. CBOW (Continuous Bag of Words):
   Input: 'the', 'sat', 'on' -&gt; Predict: 'cat'
   Faster training

Training objective (Skip-gram):
  maximize P(context | center)
  = softmax(v_context · v_center)

Famous property - Word Analogies:
  king - man + woman ≈ queen
  paris - france + italy ≈ rome

Using pre-trained embeddings:
  - Word2Vec (Google)
  - GloVe (Stanford)
  - FastText (Facebook)
```

3. Text Classification Model

3.1 Simple Text Classifier

```
print("Simple Text Classification")
print("="*70)

class TextClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_classes):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0)
        self.fc = nn.Linear(embed_dim, num_classes)

    def forward(self, x):
        # x: (batch, seq_len)
        embedded = self.embedding(x)  # (batch, seq_len, embed_dim)
        # Average pooling over sequence
        pooled = embedded.mean(dim=1)  # (batch, embed_dim)
        return self.fc(pooled)

model = TextClassifier(vocab_size=10000, embed_dim=128, num_classes=2)
print(model)

# Example: Sentiment classification
x = torch.randint(0, 10000, (4, 50))  # 4 texts, 50 tokens each
out = model(x)
print(f"\nInput shape: {x.shape}")
print(f"Output shape: {out.shape}")

print("\nPooling strategies:")
print("  - Mean: Average all embeddings")
print("  - Max: Max over each dimension")
print("  - [CLS]: Use special token (BERT-style)")
```

```
Simple Text Classification
======================================================================
TextClassifier(
  (embedding): Embedding(10000, 128, padding_idx=0)
  (fc): Linear(in_features=128, out_features=2, bias=True)
)

Input shape: torch.Size([4, 50])
Output shape: torch.Size([4, 2])

Pooling strategies:
  - Mean: Average all embeddings
  - Max: Max over each dimension
  - [CLS]: Use special token (BERT-style)
```

Summary

Key Takeaways:

- Tokenization converts text to token sequences
- Dense embeddings capture semantic similarity
- Word2Vec learns embeddings from context prediction
- Similar words have similar embeddings
- Pre-trained embeddings provide good initialization
- Pooling converts variable-length to fixed-length

Practice Exercises:

1. Implement word tokenization from scratch

2. Build vocabulary from corpus
3. Use pre-trained GloVe embeddings
4. Implement simple text classifier
5. Visualize word embeddings with t-SNE