Dimensionality Reduction for NLP

Lecture 13: PCA, t-SNE, and UMAP

PSYC 51.07: Models of Language and Communication - Week 4

Winter 2026

Today's Lecture 📋

1. 📉 **The Curse of Dimensionality**
2. 📊 **Principal Component Analysis (PCA)**
3. 🎨 **t-SNE: Stochastic Neighbor Embedding**
4. 🗺️ **UMAP: Uniform Manifold Approximation**
5. 🔍 **Comparison & Best Practices**
6. 💻 **Visualization Techniques**

Goal: Learn to visualize and explore high-dimensional embeddings

The Curse of Dimensionality 📉

The Problem:

Word Embeddings:

- Word2Vec: 300 dimensions
- GloVe: 300 dimensions
- FastText: 300 dimensions
- BERT: 768 dimensions
- GPT-3: 12,288 dimensions!

Challenges:

- Can't visualize 300D space
- Distances behave strangely

Why Dimensionality Reduction? 🎯

Visualization:

- 2D/3D plots
- Explore semantic structure
- Identify clusters
- Discover patterns
- Quality assurance

Computation:

- Faster algorithms
- Less memory
- Enable real-time systems
- Scalability

Principal Component Analysis (PCA) 📊

The classic linear dimensionality reduction method

Intuition:

- Find directions of maximum variance
- Project data onto these directions
- First PC: most variance
- Second PC: second most (orthogonal)
- And so on...

Properties:

- Linear transformation
- Preserves global structure

PCA: The Algorithm 🔧

Step-by-Step Worked Example:

Original Data (3D → 2D):

Word	x1	x2	x3
king	2.0	1.5	0.1
queen	1.8	1.6	0.2
man	1.0	0.5	0.1
woman	0.9	0.6	0.2

Step 1: Center data (subtract mean)

Step 2: Compute covariance matrix C

PCA in Practice 💻

```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt

# Assume we have word embeddings: shape (vocab_size, 300)
# For example, Word2Vec embeddings

# Standardize the data (optional but recommended)
scaler = StandardScaler()
embeddings_scaled = scaler.fit_transform(embeddings)

# Apply PCA
pca = PCA(n_components=2)  # Reduce to 2D for visualization
embeddings_2d = pca.fit_transform(embeddings_scaled)

# Check variance explained
print(f"Variance explained: {pca.explained_variance_ratio_}")
```

PCA in Practice 💻

```python
# Visualize
plt.figure(figsize=(10, 8))
plt.scatter(embeddings_2d[:, 0], embeddings_2d[:, 1], alpha=0.5)

# Annotate some words
words = ['king', 'queen', 'man', 'woman', 'cat', 'dog']
for word in words:
    idx = word_to_idx[word]
    plt.annotate(word, (embeddings_2d[idx, 0], embeddings_2d[idx, 1]))

plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Word Embeddings - PCA Projection')
plt.show()
```

...continued

PCA Limitations ⚠️

Assumes Linearity:

- Only captures linear relationships
- Word embeddings often have non-linear structure
- May miss important patterns

```
1Non-linear manifold ->  PCA struggles here!
```

Other Issues:

- **Variance ≠ Importance:**
- Preserves variance, not semantic structure
- Noisy dimensions may have high variance

t-SNE: t-Distributed Stochastic Neighbor Embedding 🎨

Non-linear dimensionality reduction for visualization

Key Idea:

- Model similarity as probability
- High-D: Gaussian similarity
- Low-D: t-distribution similarity
- Minimize divergence between them
- Preserves

Advantages:

- Beautiful visualizations

t-SNE: Key Concepts 🔍

Why t-distribution in low-D?

The Crowding Problem Illustrated:

```
High-D: 10 points can each have
        9 equidistant neighbors

      *   *   *
        * * *
      *   *   *

Low-D (2D): Can't fit 9 equidistant
            neighbors around 1 point!

Solution: t-distribution has
          heavier tails → allows
          moderately-distant points
          to spread further apart
```

t-SNE in Practice 💻

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# Apply t-SNE (can be slow for large datasets)
tsne = TSNE(
    n_components=2,      # 2D visualization
    perplexity=30,       # try 5-50
    n_iter=2000,         # iterations
    random_state=42,     # reproducibility
    verbose=1            # show progress
)

embeddings_2d = tsne.fit_transform(embeddings)

# Visualize with labels
plt.figure(figsize=(12, 10))

# Color by semantic category (if available)
```

t-SNE in Practice 💻

```

for cat, color in zip(categories, colors):
    mask = labels == cat
    plt.scatter(
        embeddings_2d[mask, 0],
        embeddings_2d[mask, 1],
        c=color,
        label=cat,
        alpha=0.6
    )

# Annotate some words
for word, idx in word_to_idx.items():
    if word in important_words:
        plt.annotate(
            word,
            (embeddings_2d[idx, 0], embeddings_2d[idx, 1])
        )
```

t-SNE in Practice 💻

```
plt.title('Word Embeddings - t-SNE Projection')
plt.show()
```

...continued

t-SNE Limitations and Caveats ⚠️

1. **Slow:** $O(n^2)$ complexity

```
# Timing comparison
n=1000:  ~10 seconds
n=5000:  ~4 minutes
n=10000: ~15 minutes
```

2. **Non-deterministic:**

```
tsne1 = TSNE(random_state=42)
tsne2 = TSNE(random_state=123)
# Different layouts!
```

3. **No out-of-sample:**

UMAP: Uniform Manifold Approximation & Projection 🗺️

Modern alternative to t-SNE (2018)

Key Advantages over t-SNE:

- $(O(n \log n)$ vs $O(n^2))$
- Preserves local and global structure
- Can transform new points
- Theoretically grounded (topology)
- Scales to millions of points
- More robust hyperparameters

When to Use:

UMAP in Practice 💻

```python
import umap
import matplotlib.pyplot as plt

# Apply UMAP
reducer = umap.UMAP(
    n_components=2,       # 2D visualization
    n_neighbors=15,       # local/global balance (try 5-50)
    min_dist=0.1,         # cluster tightness (try 0.0-0.99)
    metric='cosine',      # good for word embeddings
    random_state=42
)

embeddings_2d = reducer.fit_transform(embeddings)

# Can also transform new points! (unlike t-SNE)
new_embeddings_2d = reducer.transform(new_embeddings)

# Visualize
```

UMAP in Practice 💻

```
    embeddings_2d[:, 0],
    embeddings_2d[:, 1],
    c=cluster_labels,     # color by cluster
    cmap='Spectral',
    s=5,
    alpha=0.6
)
plt.colorbar(scatter)

# Annotate
for word in important_words:
    idx = word_to_idx[word]
    plt.annotate(
        word,
        (embeddings_2d[idx, 0], embeddings_2d[idx, 1]),
        fontsize=12
    )

```

PCA vs. t-SNE vs. UMAP 🔍

Feature	PCA	t-SNE	UMAP
Speed	Very Fast	Slow	Fast
Scalability	Excellent	Poor (<10k)	Excellent
Global structure	Yes	No	Yes
Local structure	Partial	Yes	Yes
Deterministic	Yes	No	Partial
Out-of-sample	Yes	No	Yes

Concrete Timing Comparison (10,000 word embeddings):

```
# PCA: 0.3 seconds
```

Visualization Best Practices 🎨

1. **Preprocessing:**
    - Standardize features (mean=0, std=1)

- Remove outliers (optional)
- For very large datasets: sample or use PCA first

2. **Try multiple hyperparameters:**
    - t-SNE perplexity: 5, 10, 30, 50

- UMAP n_neighbors: 5, 15, 30, 50
- Different views reveal different structure

3. **Color intelligently:**
    - By semantic category

- By cluster assignment

Interactive Visualization 💡

```
import plotly.express as px
import plotly.graph_objects as go
import pandas as pd

# Prepare data
df = pd.DataFrame({
    'x': embeddings_2d[:, 0],
    'y': embeddings_2d[:, 1],
    'word': words,
    'category': categories,
    'frequency': frequencies
})

# Create interactive plot with plotly
fig = px.scatter(
    df,
    x='x',
    y='y',
```

Interactive Visualization 💡

```python
    hover_data=['word', 'frequency'],
    title='Word Embeddings (UMAP)',
    width=1000,
    height=800
)

# Add text annotations for important words
for word in important_words:
    row = df[df['word'] == word].iloc[0]
    fig.add_annotation(
        x=row['x'],
        y=row['y'],
        text=word,
        showarrow=False,
        font=dict(size=12)
    )

# Show
```

Interactive Visualization 💡

```
# Can also save as HTML
fig.write_html('embeddings_viz.html')
```

…continued

Applications in NLP 🚀

1. Embedding Quality Check:

```
# Quick sanity check
words = ['dog', 'cat', 'fish',   # animals
          'car', 'bus', 'train'] # vehicles

# If good embeddings: two clusters!
# If bad: random scatter
```

2. Finding Polysemous Words:

```
"apple" appears in TWO clusters:
- Near: orange, banana, fruit
- Near: microsoft, google, tech

→ Word has multiple senses!
```

Case Study: Visualizing BERT Layers 🔬

Question: What do different BERT layers capture?

Approach:

1. Extract embeddings from each layer
2. Apply UMAP to each layer separately
3. Visualize and compare

Typical Findings:

- **Layer 0-2:** Syntactic (POS tags cluster)
- **Layer 3-8:** Semantic (meaning clusters)
- **Layer 9-12:** Task-specific

Insights:

Case Study: Visualizing BERT Layers 🔬

```
viz_2d = reducer.fit_transform(avg_embeddings)

# Plot
plt.figure()
plt.scatter(viz_2d[:, 0], viz_2d[:, 1], c=pos_tags)
plt.title(f'Layer {layer_idx}')
plt.show()
```

…continued

Reference: Jawahar et al. (2019). "What Does BERT Learn about the Structure of Language?"

Discussion Question 💬

What does a 2D visualization actually tell us about 300D space?

Consider:

- We compress 300 dimensions into 2
- Massive information loss (>99%)
- Different methods show different views
- Hyperparameters change the story

What we CAN conclude:

- Rough semantic groupings
- Relative proximities
- Cluster existence

Practical Tips 💡

1. **Choose the right tool:**
    - Default: UMAP (fast, balanced)

- Need speed: PCA
- Small dataset, only local: t-SNE

2. **Preprocessing matters:**
    - StandardScaler for PCA

- Consider normalization for UMAP/t-SNE
- Remove extreme outliers

3. **Try multiple settings:**
    - Run with different hyperparameters

- Compare results

Summary 🎯

What we learned today:

1. **Curse of Dimensionality:** High-D space is weird, need reduction
2. **PCA:**
    - Linear, fast, preserves global structure

- Good for preprocessing and quick exploration

3. **t-SNE:**
    - Non-linear, preserves local structure

- Beautiful visualizations but slow
- Sensitive to hyperparameters

4. **UMAP:**
    - Fast, scalable, balanced local+global

Key References 📚

Foundational Papers:

- Pearson, K. (1901). "On Lines and Planes of Closest Fit to Systems of Points in Space" (PCA)
- van der Maaten & Hinton (2008). "Visualizing Data using t-SNE"
- McInnes et al. (2018). "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction"

Applications:

- Jawahar et al. (2019). "What Does BERT Learn about the Structure of Language?"
- Coenen et al. (2019). "Visualizing and Measuring the Geometry of BERT"

Guides & Tutorials:

- Wattenberg et al. (2016). "How to Use t-SNE Effectively" (Distill)

Questions? 🙋

Next Lecture:

Cognitive Models of Semantic Representation

How do humans represent meaning?