Lecture Information

Course: CSCI E-89B: Natural Language Processing
Lecture: Lecture 7
Topic: Latent Dirichlet Allocation and Topic Modeling
Date: Fall 2024

Contents

1 Quiz Review: Autoencoders Revisited

Overview

This lecture begins with a review of autoencoder concepts before introducing topic modeling—a powerful unsupervised learning technique for discovering hidden themes in document collections.

1.1 One-Hot Encoding Disadvantages

One-Hot Encoding Problems

One-hot encoding has two major disadvantages:

1. **Sparsity**: The representation contains many zeros
2. **High dimensionality**: As a direct result of sparsity

Why sparsity is problematic:

- When computing linear combinations, you perform many operations with zeros
- $0 \times$ something contributes nothing but still requires computation
- Information is represented inefficiently
- This is precisely why we use embeddings instead

1.2 Undercomplete Autoencoders

Undercomplete Autoencoder

An autoencoder is called **undercomplete** when the dimensionality of the representation (encoding) is **lower** than the dimensionality of the input. This creates a bottleneck that forces compression.

Understanding "Undercomplete"

The terminology makes intuitive sense:

- **Complete**: 4 dimensions in $\rightarrow$ 4 dimensions in middle $\rightarrow$ 4 dimensions out
- **Undercomplete**: 4 dimensions in $\rightarrow$ 2 dimensions in middle $\rightarrow$ 4 dimensions out
- **Overcomplete**: 3 dimensions in $\rightarrow$ 7 dimensions in middle $\rightarrow$ 3 dimensions out

If you have a 3-dimensional dataset and map it to 2 dimensions, your representation is "undercomplete" because you cannot fully represent 3D data in 2D without some loss. You're taking projections rather than keeping complete information.

1.3 Stacked Autoencoders

Stacked Autoencoder

A **stacked autoencoder** (also called a deep autoencoder) has multiple hidden layers. "Stacked" by definition implies depth—at least 2 hidden layers.

- **Shallow network**: 1 hidden layer only
- **Deep network**: 2 or more hidden layers

1.3.1 Layer-wise Training

Important

Layers of stacked autoencoders do not need to be trained together. You can train them one at a time using a "sandwich" approach:

Phase 1: Train only input → hidden_1 → output (no middle layers)

Phase 2: Freeze coefficients, add hidden_2 between hidden_1 and output

Phase 3: Train middle part while keeping frozen layers fixed

Phase 4: Continue stacking more layers

1.3.2 Why Layer-wise Training?

The motivation relates to gradient descent optimization:

The Scale Problem

Weights W for different layers can be on different scales:

- W_1 (near input) and W_{200} (near output) may differ vastly in magnitude
- Cost function level curves become stretched (elliptical rather than circular)
- The gradient $-\alpha\nabla J$ points away from the true minimum
- Training may diverge or take forever

Modern solutions to the scale problem:

1. **Adam optimizer**: Adjusts gradient direction based on local curvature
2. **Batch normalization**: Normalizes signals locally, mitigating weight scale issues
3. **Shortcut connections**: Connect layers to later layers, allowing signals to bypass problematic areas

Modern Practice

Due to these techniques, the scale problem is largely solved today. People often train deep autoencoders with all layers at once. However, the phase-wise approach remains elegant and worth knowing.

1.4 Autoencoders for Sequences

Important

Autoencoders **can** be used for sequences. A sequence of words (sentence) can be compressed into a single vector (the bottleneck), which has no time component.

$$\text{Sequence } \underbrace{[x_1, x_2, \ldots, x_T]}_{\text{vectors}} \xrightarrow{\text{Encoder}} \underbrace{z}_{\text{single vector}} \xrightarrow{\text{Decoder}} \text{Reconstructed sequence}$$

1.4.1 How Small Should the Bottleneck Be?

This is a fundamental question with no universal answer:

Goal Determines Architecture

If your goal is perfect reconstruction:

- Don't squeeze at all!
- Why would you compress if you need to recover everything?
- Just use input = output directly

If your goal is feature extraction for downstream tasks:

- Use the encoder output as input to a classifier
- The optimal bottleneck size depends on classification performance
- Iterate: try 2D, 5D, 10D representations and evaluate which works best

Practical Consideration

If you have millions of input features, feeding them directly to a classifier creates millions of parameters. You may want to compress first to:

- Reduce parameter count
- Avoid getting stuck in local minima
- Make optimization tractable

2 Introduction to Topic Modeling

Overview

Topic modeling is an **unsupervised learning** approach for discovering hidden themes in document collections. Unlike clustering, documents can belong to **multiple topics** with different proportions.

2.1 Motivation: Why Not Clustering?

Traditional Clustering

Standard clustering algorithms (K-means, hierarchical) assign each data point to **exactly one cluster**:

- Students belong to "students" cluster
- Retired people belong to "retired" cluster
- Points don't overlap between clusters

The problem with text documents:

Documents Are Different

In text, topics naturally **overlap**:

- A news article may start discussing economics, shift to politics, and mention artificial intelligence
- The same document contains multiple themes
- Hard assignment to one cluster loses important information

Document as Topic Mixture

Consider Document 2 in a corpus:

- Topic 1 (Economics): 60%
- Topic 2 (Politics): 30%
- Topic 3 (AI): 10%

This soft assignment captures the document's multi-topical nature far better than forcing it into one category.

2.2 Topic Modeling Methods

Three main approaches for topic modeling:

Method	Type	Characteristics
LDA	Probabilistic	Assumes stochastic text generation
NMF	Deterministic	Matrix factorization approach
STM	Probabilistic	LDA + covariates (metadata)

Why Social Scientists Love Topic Models

Topic modeling is popular in political science, government departments, and social sciences:

- Extract information from large text corpora computationally
- Discover biases and patterns

- Correlate topics with covariates (gender, source, etc.)

2.3 Latent Topics

Latent Topics

Topics are called "latent" because:

- We don't define topics upfront (e.g., "economics")
- The model discovers topics from data
- We only assign labels **after** processing
- Labels come from examining top-weighted documents/words per topic

Important

The number of topics is a **hyperparameter** you must specify beforehand. If you say 3 topics, the model will find exactly 3 topics—it won't tell you the "true" number.

3 Maximum Likelihood Estimation

Overview

Before diving into LDA, we need to understand **maximum likelihood estimation (MLE)**—the framework used to recover model parameters from observed data.

3.1 The MLE Framework

Maximum Likelihood Estimation

MLE finds parameters that maximize the probability of observing the data we actually observed:

$$\hat{\theta} = \arg\max_{\theta} P(\text{data}|\theta)$$

Simple Example: Normal Distribution

Suppose we observe data and create a histogram. We assume data comes from a normal distribution with unknown μ and σ^2.
Step 1: Try parameters (Model A): μ far left of data

- Probability of observing our data given Model A is very small
- Likelihood ≈ 0

Step 2: Try parameters (Model B): μ closer to data center

- Probability is higher
- Likelihood increases

Step 3: Try parameters (Model C): μ at data mean, appropriate σ^2

- Probability is highest
- This is our MLE solution!

Step 4: Try parameters (Model D): Overshoot μ

- Probability decreases again

Conceptual diagram: Scanning through parameter space to find maximum likelihood

Important

MLE is incredibly powerful:

- Works with many parameters (dozens or more)
- Used in time series, LDA, STM, and countless applications
- Very natural approach: find parameters that make observed data most probable

4 Latent Dirichlet Allocation (LDA)

Overview

LDA models each document as a **mixture of topics**, where each topic is a distribution over words. Unlike clustering, LDA provides soft (probabilistic) topic assignments.

4.1 The Generative Model

LDA assumes text is generated by the following process:

LDA Text Generation Process

For document m:

Step 1: Choose number of words $N \sim \text{Poisson}(\xi)$

Step 2: Choose topic proportions $\theta_m \sim \text{Dirichlet}(\alpha)$

Step 3: For each word $n = 1, \ldots, N$:

(a) Choose topic $z_n \sim \text{Multinomial}(\theta_m)$
(b) Choose word $w_n \sim \text{Multinomial}(\beta_{z_n})$

4.2 Understanding Each Component

4.2.1 Poisson Distribution for Document Length

Poisson Distribution

The Poisson distribution models count data:

$$P(N = k) = \frac{\xi^k e^{-\xi}}{k!}$$

where ξ is both the mean and variance. If documents average 25 words, $\xi \approx 25$.

4.2.2 Dirichlet Distribution for Topic Proportions

Dirichlet Distribution

The Dirichlet distribution produces vectors of probabilities that sum to 1. For K topics:

$$\theta_m = (\theta_{m,1}, \theta_{m,2}, \ldots, \theta_{m,K}) \quad \text{where } \sum_k \theta_{m,k} = 1$$

Example for 3 topics: $\theta_m = (0.6, 0.3, 0.1)$ means:

- 60% Topic 1 (Economics)
- 30% Topic 2 (Politics)
- 10% Topic 3 (AI)

Dirichlet as Generalized Beta

- For 2 topics: Dirichlet reduces to the Beta distribution
- For $K > 2$ topics: Dirichlet is the natural generalization
- Parameter α (vector) controls the shape of the distribution

4.2.3 Multinomial Distribution for Topic and Word Selection

Multinomial Selection

Given probabilities, multinomial sampling selects one category:

- $z_n \sim \text{Multinomial}(\theta_m)$: Select topic for word n
- $w_n \sim \text{Multinomial}(\beta_{z_n})$: Select word from chosen topic's vocabulary distribution

Concrete Word Generation

Given $\theta_m = (0.1, 0.2, 0.6)$ for three topics:

1. Draw topic: Most likely Topic 3 (60%), but could be Topic 2 (20% chance)
2. Suppose Topic 2 is selected
3. Draw word from β_2: Each topic has its own word distribution

4. Word "maximization" is selected from Topic 2's distribution

This is repeated independently for each word position.

4.3 Key Assumptions of LDA

Bag of Words Assumption

LDA assumes **no word order**:

- Words are generated independently
- "maximization" doesn't depend on what came before
- Shuffling words in a document doesn't change its LDA representation
- Generated text wouldn't be grammatical—that's not the point!

Important

LDA is not for generating readable text. It's for:

- Discovering what topics a document discusses
- Finding topic proportions for each document
- Identifying words associated with each topic

4.4 The EM Algorithm

Expectation-Maximization (EM)

LDA uses the EM algorithm because topic assignments z_n are **latent variables** (not observed):

1. **E-step**: Estimate expected values of latent variables z_n given current parameters
2. **M-step**: Maximize likelihood given these expected values
3. Iterate until convergence

Why EM?

We observe documents (sequences of words) but not:

- Which topic each word came from (z_n)
- The true topic proportions (θ_m)
- The topic-word distributions (β_k)

EM handles this missing information elegantly.

5 LDA Implementation in Python

5.1 Using scikit-learn

```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

# Example documents
documents = [
    "Cats are wonderful pets",
    "Cats and dogs are popular animals",
    "Dogs enjoy long walks",
    "Walks in the park are relaxing",
    # ... more documents
]

# Create document-term matrix
vectorizer = CountVectorizer(stop_words='english')
X = vectorizer.fit_transform(documents)

# Fit LDA
lda = LatentDirichletAllocation(
    n_components=2,       # Number of topics
    random_state=42       # For reproducibility
)
lda.fit(X)

# Get topic-document distribution
doc_topic_dist = lda.transform(X)
print(doc_topic_dist)
# Output: [[0.05, 0.95], [0.06, 0.94], [0.93, 0.07], ...]
```

Listing 1: LDA with scikit-learn

Interpreting Results

Output $[0.05, 0.95]$ means:

- 5% contribution from Topic 1
- 95% contribution from Topic 2

Since this document is mostly Topic 2, and it's about cats, Topic 2 is likely the "cats" topic.

5.2 Extracting Top Words per Topic

```python
def display_topics(model, feature_names, num_top_words=5):
    for topic_idx, topic in enumerate(model.components_):
        top_words_idx = topic.argsort()[:-num_top_words-1:-1]
        top_words = [feature_names[i] for i in top_words_idx]
        print(f"Topic {topic_idx}: {', '.join(top_words)}")

feature_names = vectorizer.get_feature_names_out()
display_topics(lda, feature_names)

# Output:
# Topic 0: dogs, enjoy, long, walks, exploring
# Topic 1: cats, purr, love, climb, trees
```

Listing 2: Display top words for each topic

5.3 Document-Based Topic Interpretation

Important

A better way to understand topics: look at documents with highest topic prevalence, not just top words.

```python
# For each topic, find documents with highest prevalence
for topic_idx in range(n_topics):
    # Sort documents by topic prevalence
    top_docs = doc_topic_dist[:, topic_idx].argsort()[::-1][:2]

    print(f"\nTopic {topic_idx} - Top documents:")
    for doc_idx in top_docs:
        prevalence = doc_topic_dist[doc_idx, topic_idx]
        print(f"  [{prevalence:.2%}] {documents[doc_idx]}")
```

Listing 3: Find most representative documents

Why Document-Based Interpretation?

Looking at actual documents is more reliable than top words because:

- Top words may be ambiguous or share meanings
- Documents provide context
- 95% prevalence means the document is almost entirely about that topic
- Reading the document tells you definitively what the topic represents

6 Non-negative Matrix Factorization (NMF)

Overview

NMF is a **deterministic** alternative to LDA. It uses linear algebra rather than probabilistic modeling to decompose documents into topics.

6.1 The Matrix Factorization Idea

NMF Decomposition

Given document-term matrix V (documents $\times$ vocabulary):

$$V \approx W \cdot H$$

where:

- W: Document-topic matrix (documents $\times$ topics)
- H: Topic-word matrix (topics $\times$ vocabulary)
- All entries in W and H are ≥ 0 (non-negative)

6.2 Matrix Multiplication Review

Matrix Multiplication Example

$$W = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad H = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

$$V = W \cdot H = \begin{pmatrix} 1 \cdot 1 + 2 \cdot 0 & 1 \cdot 1 + 2 \cdot 0 & 1 \cdot 0 + 2 \cdot 2 \\ 3 \cdot 1 + 4 \cdot 0 & 3 \cdot 1 + 4 \cdot 0 & 3 \cdot 0 + 4 \cdot 2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 4 \\ 3 & 3 & 8 \end{pmatrix}$$

Rules:

- $(m \times n) \cdot (n \times p) = (m \times p)$
- Element (i, j) = dot product of row i from first matrix and column j from second

6.3 NMF for Topic Modeling

Concrete NMF Example

Three documents with vocabulary [cats, dogs, bark, purr, growl]:

- Doc 1: “cats meow” → [1, 0, 0, 1, 0]
- Doc 2: “dogs bark” → [0, 1, 1, 0, 0]
- Doc 3: “cats purr dogs growl” → [1, 1, 0, 1, 1]

$$V = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \end{pmatrix}$$

NMF finds:

$$W = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0.5 & 0.5 \end{pmatrix}, \quad H = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Interpretation:

- Topic 1: cats-related (cats, purr)
- Topic 2: dogs-related (dogs, bark, growl)
- Doc 3: 50% Topic 1, 50% Topic 2

6.4 NMF vs LDA

Aspect	LDA	NMF
Approach	Probabilistic	Deterministic
Algorithm	EM (iterative)	Matrix factorization
Reproducibility	Stochastic (varies)	Deterministic (same result)
Interpretability	Often better	Harder to interpret W values
Speed	Slower	Faster
Constraints	Probabilities sum to 1	Only non-negativity

NMF Interpretation Challenge

NMF's W matrix entries are just non-negative numbers, not probabilities:

- Values like 2.0 and 5.0 are valid
- Harder to say "60% Topic 1"
- Requires normalization for probability interpretation

6.5 NMF Implementation

```
from sklearn.decomposition import NMF
from sklearn.feature_extraction.text import CountVectorizer

# Same document-term matrix X from before
nmf = NMF(n_components=2, random_state=42)
W = nmf.fit_transform(X)  # Document-topic matrix
H = nmf.components_         # Topic-word matrix

# Display top words per topic
for idx, topic in enumerate(H):
    top_words = [feature_names[i] for i in topic.argsort()[-5:]]
    print(f"Topic {idx}: {top_words}")
```

Listing 4: NMF in Python

7 Choosing the Number of Topics

Overview

Selecting the optimal number of topics is crucial. Two key metrics help: **coherence** (within-topic word relatedness) and **exclusivity** (between-topic word distinctiveness).

7.1 Coherence

Topic Coherence

Coherence measures how related the top words within a topic are to each other. Higher coherence means the topic's top words frequently co-occur in documents.

Intuition: If Topic 1's top words are [dogs, walks, fetch, leash, park], these words should appear together in documents more often than random word pairs.

Computing Coherence (Simplified)

For top words in a topic, compute co-occurrence:

$$\text{Coherence} \propto \sum_{i<j} \log \frac{P(w_i, w_j) + \epsilon}{P(w_i) \cdot P(w_j)}$$

This is essentially a log-transformed correlation measure.

7.2 Exclusivity

Topic Exclusivity

Exclusivity measures how distinct topics are from each other. High exclusivity means top words in one topic don't appear as top words in other topics.

Exclusivity Calculation (Simplified)

For word w in topic k:

$$\text{Exclusivity}(w, k) = \frac{P(w|\text{topic } k)}{\sum_{k'} P(w|\text{topic } k')}$$

If "dogs" has high probability only in Topic 1 and low in others, it has high exclusivity for Topic 1.

7.3 The Coherence-Exclusivity Trade-off

Trade-off

Coherence and exclusivity often compete:

- More topics → Higher exclusivity but potentially lower coherence
- Fewer topics → Higher coherence but topics may overlap

Important

Selection strategy:

1. Compute coherence and exclusivity for different numbers of topics
2. Standardize both metrics (z-scores)
3. Plot: x-axis = exclusivity, y-axis = coherence
4. Choose model closest to top-right corner (high both)

```python
from gensim.models.coherencemodel import CoherenceModel

# After training LDA model
coherence_model = CoherenceModel(
    model=lda_gensim,
    texts=tokenized_docs,
    dictionary=dictionary,
    coherence='c_v',      # Coherence type
    topn=20               # Top 20 words per topic
)
coherence_score = coherence_model.get_coherence()
print(f"Coherence: {coherence_score:.4f}")
```

Listing 5: Coherence calculation with gensim

7.4 Multiple Runs

Stochastic Nature of LDA

LDA results vary between runs due to:

- Random initialization
- EM algorithm finding different local optima
- Topic labeling is arbitrary (Topic 1 in run A might be Topic 2 in run B)

Best practice: Run multiple times with different seeds, evaluate coherence/exclusivity, select best run.

8 LDA Implementation in R

Overview

R has excellent support for topic modeling, especially for Structural Topic Modeling (STM). Learning basic R is worthwhile for NLP research.

8.1 Setting Up R

1. Download R from `https://cran.r-project.org/`
2. Download RStudio from `https://posit.co/`
3. Install R first, then RStudio (so RStudio finds R automatically)

8.2 Basic R Syntax for LDA

```r
# Install and load packages
install.packages("topicmodels")
library(topicmodels)

# Create documents
documents <- c(
  "cats are wonderful pets",
  "cats and dogs are popular",
  "dogs enjoy long walks",
  # ... more documents
)

# Preprocessing
corpus <- Corpus(VectorSource(documents))
corpus <- tm_map(corpus, tolower)
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeWords, stopwords("english"))

# Create Document-Term Matrix
dtm <- DocumentTermMatrix(corpus)

# Fit LDA
lda_model <- LDA(dtm, k = 2, control = list(seed = 42))

# Get top terms per topic
```

```r
terms(lda_model, 5)
```

Listing 6: LDA in R

8.3 R Markdown for Reports

RMarkdown Files (.Rmd)

Similar to Jupyter notebooks:

- Mix code and text
- Code in "chunks" (like cells)
- `Ctrl+Enter` runs current line
- `Knit` creates HTML/PDF report
- Use `#` for sections, `##` for subsections

9 Practical Applications

9.1 What Can You Do with Topic Models?

1. **Document Classification**: Assign documents to dominant topics
2. **Search**: Find documents about specific topics
3. **Trend Analysis**: Track topic prevalence over time
4. **Bias Detection**: Correlate topics with metadata (gender, source)
5. **Summarization**: Understand what a corpus is "about"

9.2 Real-World Example: Student Evaluations

Analyzing Student Evaluations

Study with 1 million student evaluations:

- Extracted 11 topics from evaluation text
- Topics: "caring instructor", "interesting lectures", "good feedback", etc.
- Correlated with instructor gender

Findings:

- Female instructors: more mentions of "caring", "facilitates discussion", "nice feedback"
- Male instructors: more mentions of "humor", "interesting", "relevant"
- This pattern persisted after controlling for department and course type
- Suggests systematic bias in how students perceive instructors

9.3 Comparing Across Departments

Topic Variation by Division

Same student evaluation study:

- **Sciences**: "explains complex concepts effectively"
- **Humanities**: "facilitates effective discussions"
- **Freshman seminars**: "positive timely feedback"

These differences reflect genuine pedagogical differences across disciplines.

10 Structural Topic Modeling (STM) Preview

Overview

STM extends LDA by incorporating **covariates**—document-level metadata that can affect topic prevalence and content.

LDA vs STM

- **LDA**: Documents have topic proportions, but ignores metadata
- **STM**: Topic proportions can depend on covariates (author gender, publication source, date, etc.)

When STM Helps

Analyzing news articles with known sources:

- LDA: Discovers topics, then you manually correlate with sources
- STM: Directly models how source affects topic distribution
- STM produces more accurate topics by using all available information

Software Note

STM is primarily implemented in R (`stm` package). Python implementations exist but are unofficial. For serious STM work, learn R.

11 One-Page Summary

Summary

Topic Modeling discovers hidden themes in document collections.
Why Not Clustering?

- Documents contain multiple topics (soft assignment)
- Clustering forces hard assignment to one cluster

LDA (Latent Dirichlet Allocation):

- Probabilistic model: documents are mixtures of topics
- Topics are distributions over words
- Generative process: choose topic proportions, then for each word, choose topic then word
- Uses EM algorithm for parameter estimation
- Number of topics is a hyperparameter

NMF (Non-negative Matrix Factorization):

- Deterministic approach: $V \approx W \cdot H$
- Faster but harder to interpret
- No probabilistic interpretation

Choosing Number of Topics:

- **Coherence**: Are top words related? (higher = better)
- **Exclusivity**: Are topics distinct? (higher = better)
- Balance both; maximize average of standardized scores

Interpreting Topics:

- Look at top words per topic
- Better: examine documents with highest topic prevalence
- Assign human-readable labels after analysis

Key Formulas:

$$\theta_m \sim \text{Dirichlet}(\alpha) \quad \text{(topic proportions)}$$
$$z_n \sim \text{Multinomial}(\theta_m) \quad \text{(topic selection)}$$
$$w_n \sim \text{Multinomial}(\beta_{z_n}) \quad \text{(word selection)}$$
$$V \approx W \cdot H \quad \text{(NMF decomposition)}$$

12 Glossary

Key Terms

- **LDA**: Latent Dirichlet Allocation—probabilistic topic model
- **NMF**: Non-negative Matrix Factorization—deterministic topic model
- **STM**: Structural Topic Modeling—LDA with covariates
- **Dirichlet distribution**: Produces probability vectors summing to 1
- **Latent topic**: Hidden theme discovered from data (not predefined)
- **Topic proportions**: How much each topic contributes to a document

- **Coherence**: Metric measuring word co-occurrence within topics
- **Exclusivity**: Metric measuring distinctiveness between topics
- **EM algorithm**: Expectation-Maximization for latent variable models
- **MLE**: Maximum Likelihood Estimation—find parameters maximizing data probability
- **Undercomplete**: Bottleneck dimension < input dimension
- **Stacked autoencoder**: Deep autoencoder with multiple hidden layers
- **Bag of words**: Document representation ignoring word order
- **Document-term matrix**: Matrix of word counts (documents × vocabulary)