Autocomplete & Language Models

Word Embeddings

Word2Vec, GloVe & Neural Language Models

SESSION 06

Contents

Review: Autocorrect & HMM

Autocorrect System

Edit Distance (Levenshtein)

```
D[i,j] = min(D[i-1,j]+1, D[i,j-1]+1, D[i-1,j-1]+cost)
```

HMM & POS Tagging

HMM Parameters: λ = (A, B, π)

A: Transition | B: Emission | π: Initial

Noisy Channel Model

$$\hat{w} = \text{argmax}\, P(w) \times P(x|w)$$

Language Model Error Model

Viterbi Algorithm

Dynamic Programming cho sequence decoding

```
v[t,j] = max(v[t-1,i] × a[i,j]) × b[j,o[t]]
```

Hôm nay: Language Models & Word Embeddings →

PART 1

Language Models

N-gram Models & Autocomplete

Unigram, Bigram, Trigram

Smoothing Techniques

Autocomplete System

What is a Language Model

Definition

A Language Model (LM) is defined as a probabilistic model that assigns probabilities to sequences of words. It identifies which sentences have a higher 'likelihood of occurrence' within natural language

Core Task

$$P(W) = P(w_1, w_2, \ldots, w_n)$$

Prob of a sequence

Insight

P("The cat sat") >> P("Sat cat the")

LM captures grammar, meaning, world knowledge

Applications

Autocomplete
"I want to eat ___" → pizza, lunch, dinner

Speech Recognition
Choose transcription that has highest prob

Machine Translation
Choose the most fluent transcription

Spelling Correction
Noisy channel: argmax P(w) × P(x|w)

Text Generation
ChatGPT, GPT-4, Claude, ...

Chain Rule of Probability

CHAIN RULE

$$P(w_1, w_2, \ldots, w_n) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1,w_2) \times \ldots \times P(w_n|w_1,\ldots,w_{n-1})$$

$$= \prod P(w_k \mid w_1, \ldots, w_{k-1})$$

📝 Example: "The students opened their books"

P(The)

P(students | The)

P(opened | The students)

P(their | The students opened)

P(books | The students opened their)

Problem

More history length → harder to estimate prob

We don't have enough data to count long sequences

Solution: Markov Assumption

Limit history length

$P(w_n|w_1 \ldots w_{n-1}) \approx P(w_n|w_{n-k} \ldots w_{n-1})$

N-gram Models

Unigram (n=1)

$P(w_n|w_1...w_{n-1}) \approx P(w_n)$

Don't have context – depend on word frequency

Bigram (n=2)

$P(w_n|w_1...w_{n-1}) \approx P(w_n|w_{n-1})$

Consider 1 previous word

Trigram (n=3)

$P(w_n|w_1...w_{n-1}) \approx P(w_n|w_{n-2}, w_{n-1})$

Consider 2 previous words - balance accuracy vs sparsity

N-gram (general)

$P(w_n|w_1...w_{n-1}) \approx P(w_n|w_{n-n+1}...w_{n-1})$

4-gram, 5-gram: need more data

Maximum Likelihood Estimation

$$P(w_n|w_{n-1}) = C(w_{n-1}, w_n) / C(w_{n-1})$$

C(.) = count trong training corpus

Example: Bigram

Corpus: "I want to eat Chinese food"

Trade-off

Larger n → More context → Better accuracy

Larger n → More sparse data → Less reliable

Bigram Example

Training Corpus

"I am Sam"

"Sam I am"

"I do not like green eggs and ham"

Unigram Counts C(w)

Word	Count	Word	Count	Word	Count
I	3	like	1	and	1
am	2	green	1	ham	1
Sam	2	eggs	1	SUM	14
do	1	not	1		

Bigram Counts C(w_{n-1}, w_n)

Bigram	Count	Bigram	Count
(I, am)	2	(not, like)	1
(I, do)	1	(like, green)	1
(am, Sam)	1	(green, eggs)	1
(Sam, I)	1	(eggs, and)	1
(do, not)	1	(and, ham)	1

Bigram Probabilities

P(am | I) = C(I, am) / C(I)

= 2 / 3 = 0.67

P(do | I) = C(I, do) / C(I)

= 1 / 3 = 0.33

P(Sam | am) = C(am, Sam) / C(am)

= 1 / 2 = 0.50

Sentence Probability

P("I am Sam") = P(I) × P(am|I) × P(Sam|am)

= (3/14) × (2/3) × (1/2) = 0.071

Start & End Tokens

Why Special Tokens?

To help the model recognize sentence boundaries

Allows to calculate P(first word) and knows to stop generating

Special Tokens

<s>
Start of Sentence

</s>
End of Sentence

We can use: BOS/EOS, [START]/[END], [CLS]/[SEP]

Note for N-gram

Trigram need 2 <s> tokens at the beginning to calculate $P(w_1|<s>,<s>)$

Example with Bigram

Original sentence:

```
"I love NLP"
```

With special tokens:

```
<s> I love NLP </s>
```

Bigram probability:

P(sentence) = P(I|<s>) × P(love|I) × P(NLP|love) × P(</s>|NLP)

Benefits

- Can calculate P(first word)
- Model know when a sentence ends
- Necessary for text generation

Sparsity Problem

Core Problem

Many N-gram combinations do not appear in training data

→ Count = 0 → Probability = 0

Scale of the Problem

V = 50,000 words

V^2 = 2.5 billion possible pairs

V^3 = 125 trillion combinations!

Catastrophic Consequence

P("The cat jumped") = ... × P(jumped|cat) × ...

If "cat jumped" have not seen → P = 0 → all sentences = 0!

Solutions: Smoothing

1. Add-k Smoothing (Laplace)

Thêm k vào mọi count

2. Backoff

Fallback to lower-order N-gram

3. Interpolation

Weighted average of N-grams

4. Kneser-Ney

State-of-the-art cho N-gram LMs

Idea

"Steal" probability mass from seen events

Redistribute to unseen events

Add-k Smoothing (Laplace)

📐 Formula

$$P(w_n|w_{n-1}) = (C(w_{n-1}, w_n) + k) / (C(w_{n-1}) + k \times V)$$

k = smoothing parameter (Ussually k=1 or k<1)

V = vocabulary size

Example: k = 1

Without smoothing:

P(jumped|cat) = 0/100 = **0**

With Add-1 (V = 10,000):

P(jumped|cat) = (0+1)/(100+10000) = **0.0001**

Advantages

- Simple to implement
- No zero probabilities
- Easy to understand

Disadvantages

- Add-1 is to aggressive with large V
- Moves too much probability mass
- Not good for text applications

Add-k & Best Practice

Use k < 1: k=0.5, k=0.1, k=0.01

Add-k → classification | LM → Interpolation

Backoff & Interpolation

Backoff

If N-gram does not exist → use (N-1)-gram

Algorithm:

```
if C(w_{n-2}, w_{n-1}, w_n) > 0:
  return P_trigram
elif C(w_{n-1}, w_n) > 0:
  return α × P_bigram
else:
  return β × P_unigram
```

Example

P(food | Chinese, eat)?

If "Chinese eat food" have not seen → fallback to P(food|eat)

Interpolation

Always mix all N-gram levels

Formula:

$$\hat{P}(w_n|w_{n-2},w_{n-1}) = \lambda_1 \times P(w_n|w_{n-2},w_{n-1}) + \lambda_2 \times P(w_n|w_{n-1}) + \lambda_3 \times P(w_n)$$

Constraint: $\lambda_1 + \lambda_2 + \lambda_3 = 1$

Why Better?

- Always uses all available information
- No zero probabilities
- λ weights learned from held-out data

Typical λ values: λ_1=0.6, λ_2=0.3, λ_3=0.1

Autocomplete System

Autocomplete Pipeline

Input Processing
Get context (previous N-1 words)

Candidate Generation
Find all words following context

Probability Scoring
Calculate P(word|context) for each

Ranking & Return
Return top-k suggestions

Example: Trigram Autocomplete

User types:

`"I want to |"`

Context = ("want", "to")

eat

go

buy

PART 2

Perplexity

Evaluating Language Models

Definition & Intuition

Mathematical Formula

Interpretation

What is Perplexity?

Definition

Perplexity (PP) is a metric for evaluating the quality of a language model. It measures how 'surprised' the model is when it encounters test data.

Formula

$$PP(W) = P(w_1, w_2, \ldots, w_n)^{(-1/N)}$$

$$= \sqrt[N]{(1 / P(W))}$$

N = total number of words in test set

Log Form (numerical stability)

$$PP = \exp(-1/N \times \Sigma \log P(w_i|context))$$

Intuition: "Weighted Average Branching Factor"

Perplexity = The average number of words the model must 'choose' from at each position.

Model "confused" between 50 words

Model "confused" betwwen 500 words

Lower is Better!

Low PP = Model predict better

PP = 1 → Perfect prediction (impossible)

Perplexity: Example

Test Sentence (Bigram)

"I want to eat"

N = 4 words (excluding <s>)

Bigram Probabilities từ LM:

P(I | <s>)	= 0.20
P(want | I)	= 0.30
P(to | want)	= 0.65
P(eat | to)	= 0.10

Step-by-Step Calculation

Step 1: Joint probability

P(W) = 0.20 × 0.30 × 0.65 × 0.10

= 0.0039

Step 2: Inverse probability

1/P(W) = 1/0.0039 = 256.41

Step 3: N-th root (N=4)

PP = $\sqrt[4]{256.41}$

= 4.0 ✓

Interpretation

PP = 4 means that the model "confused" around 4 words at each position.

→ It is good for a simple bigram model!

Log form: log PP = -1/N × Σ log P = more stable

PART 3

Word Embeddings

Distributional Semantics & Dense Representations

From Sparse to Dense

Semantic Similarity

Word Analogies

Distributional Semantics

Core Idea

"You shall know a word by the company it keeps"

— J.R. Firth, 1957

Distributional Hypothesis

Words with similar meanings will appear in similar contexts.

Context of "dog":

"The ***dog*** barks", "I pet my ***dog***", "The ***dog*** ran"

Context of "cat":

"The ***cat*** meows", "I pet my ***cat***", "The ***cat*** ran"

→ Similar contexts → Similar meanings!

Evolution of Word Representations

1. One-Hot Encoding

[0, 0, 1, 0, 0, ..., 0] - Sparse, no similarity

2. TF-IDF / Co-occurrence

Count-based, still high-dimensional

3. Dense Embeddings (Word2Vec, GloVe)

[0.2, -0.5, 0.8, ...] - Low-dim, captures semantics

4. Contextualized (BERT, GPT)

Different vector for each context!

Today: Word2Vec & GloVe (Static Embeddings)

Word Embedding Properties

Semantic Similarity

Similar words → Nearby vectors in space

Cosine Similarity:

sim(king, queen) = 0.85

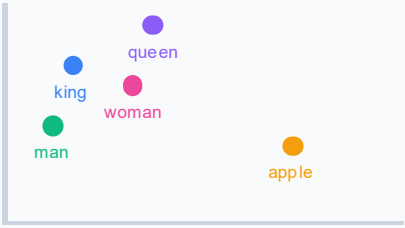
sim(king, apple) = 0.12

Word Analogies

Semantic relationships are encoded as vector operations!

$$\text{king} - \text{man} + \text{woman} \approx \text{queen}$$

Vector Space Visualization



Gender direction: man → woman ≈ king → queen

Benefits of Word Embeddings ??

PART 4

Word2Vec

Neural Word Embeddings

Word2Vec: Introduction

What is Word2Vec?

Developed by **Mikolov et al. (2013)** at Google

A family of neural network models that learn dense word vectors from large unlabeled text corpora.

Core Idea

"Predict a word from its context, or predict context from a word"

→ The hidden layer weights become the word embeddings!

Key insight: Self-supervised learning

No labeled data needed - the text itself provides supervision!

Two Architectures

1 **CBOW**

Continuous **B**ag **O**f **W**ords

Context → Target word

"The cat sat on the ___" → "mat"

2 **Skip-gram**

Inverse of CBOW

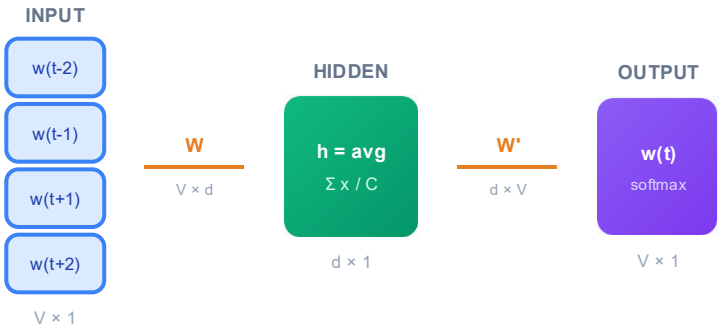
Target word → Context

"mat" → "cat", "sat", "on", "the"

Skip-gram: works better for rare words | **CBOW:** faster, better for frequent words

CBOW Architecture

Neural Network Architecture



V = vocabulary size (10K-1M) | d = embedding dimension (50-300) | C = context window size

Training Process

1. One-hot encode context words
2. Lookup embeddings from W
3. Average to get hidden state h
4. Softmax over W' × h
5. Cross-entropy loss + backprop

Example

Context: "the cat ___ on mat"

Target: "sat"

Window = 2: [the, cat, on, mat] → sat

After training

Matrix W is word embeddings!

CBOW: Mathematical Formulation

Forward Pass

1. Hidden layer (average of context embeddings)

$$h = (1/C) \times \Sigma WTxc$$

2. Output scores (dot product with all word vectors)

$$u = W' \times h$$

3. Softmax probability

$$P(wt|context) = \exp(ut) / \Sigma \exp(uj)$$

Objective: Maximize Log-Likelihood

$$J = (1/T) \times \Sigma \log P(wt|wt\text{-}c,...,wt+c)$$

T = total training examples, c = context window size

Intuition: What does training optimize?

- Words appearing in similar contexts → **similar embeddings**
- High dot product with true target → increase probability
- Push apart vectors of unrelated words

Softmax Bottleneck

Tính softmax over V words rất expensive!

V = 1,000,000 → 1M operations per word!

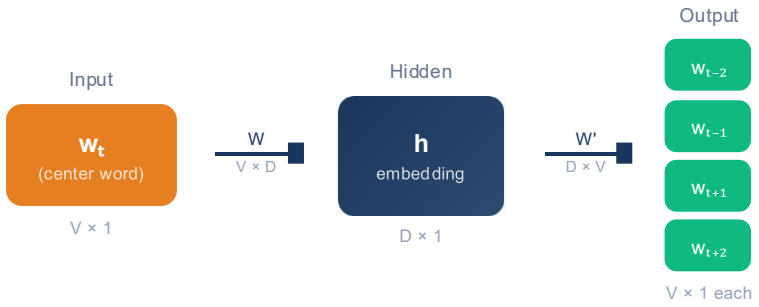
→ Solution: Negative Sampling hoặc Hierarchical Softmax

Key Parameters

d: 50-300 dims | **c:** 2-10 window | **lr:** 0.025

Skip-gram Architecture

Architecture: Center → Context



Insight: Predict MULTIPLE context words from ONE center word. Better for rare words!

Example (window = 2)

Sentence: "The quick brown fox jumps"

Input: **"brown"** → Predict: The, quick, fox, jumps

Training Objective

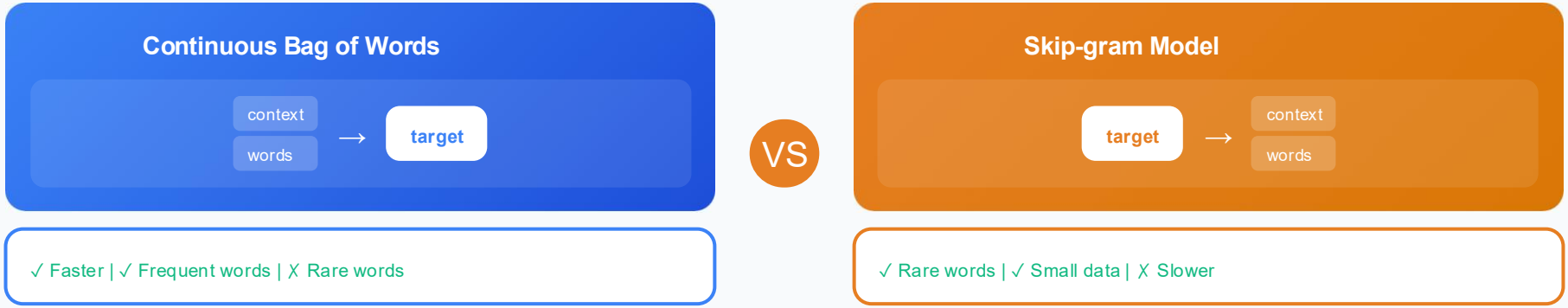
$$\max \sum t \sum -c \leq j \leq c, j \neq 0 \log P(wt+j|wt)$$

Maximize probability of context words given center word over all positions

Skip-gram Strengths:

- Works well with small data
- Better for rare words
- More training examples per word

CBOW vs Skip-gram



CBOW: large corpus | Skip-gram: small corpus + rare words

The Softmax Challenge

Standard Softmax

$$P(w|w) = \frac{\exp(u \cdot v)}{\sum \exp(u \cdot v)}$$

u_o
output vector

v_c
center vector

The Bottleneck

Denominator requires summing over **ALL vocabulary words**:

Solutions to Speed Up

Negative Sampling

Instead of all V words, sample k negative words (k = 5-20). Binary classification instead of V-way softmax.

Hierarchical Softmax

Build binary tree with words at leaves. Path length = O(log V) instead of O(V).

Negative Sampling

Idea

Instead of predicting probability over entire vocabulary, convert to **binary classification**:

✓ **Positive**
Real context pairs

✗ **Negative**
Random "fake" pairs

Example: "The cat sat on the mat"

Objective Function

$$J = \log \sigma(u_o^T v_c) + \sum_{k=1}^{K} \log \sigma(-u_k^T v_c)$$

Negative Sampling Distribution

Sample negatives based on word frequency (raised to 3/4 power):

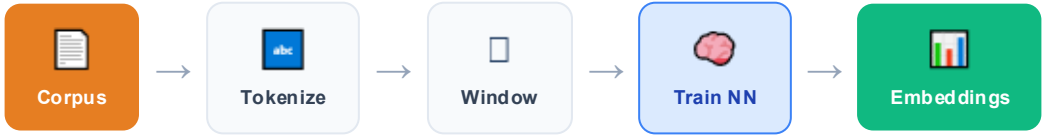
$$P(w) = f(w) / \sum f(w)$$

The 3/4 power smooths the distribution — less bias towards very frequent words

Hyperparameters

✅ **Benefit:** Train on ~15 binary classifiers instead of 100,000-way softmax!

Training Process



Prepare Corpus

Tokenize, lowercase, remove rare words (freq < 5)

Generate Training Pairs

Slide window over text, create (center, context) pairs

Initialize Embeddings

Random init W (V×D) and W' (D×V)

Forward Pass

Compute predictions using current weights

Compute Loss & Gradients

Neg sampling loss, backprop gradients

Update & Repeat

SGD update, repeat for millions of pairs

✅ Final Output

Word Embedding Matrix W:

```
[[0.12, -0.34, ...], # "the"
[0.45,  0.23, ...], # "cat"
...]
```

Shape: (vocab_size × embed_dim)

Use W or average(W, W')

PART 5

GloVe

Global Vectors for Word Representation

Co-occurrence Matrix

Weighted Least Squares

Global Statistics

GloVe: Introduction

What is GloVe?

Global **Ve**ctors — an unsupervised learning algorithm that combines:

Global Statistics
Matrix factorization
(like LSA)

+

Local Context
Window-based learning
(like Word2Vec)

Insight

Word meanings are encoded in **co-occurrence statistics**. The ratio of co-occurrence probabilities captures semantic relationships:

$$P(\text{solid} \mid \text{ice}) / P(\text{solid} \mid \text{steam}) >> 1$$

This ratio reveals "ice" is related to "solid"

Word2Vec vs GloVe Approach

Word2Vec (Predictive)

Learns embeddings by predicting context words. Slides window over corpus → many training pairs.

GloVe (Count-based)

First collects global co-occurrence counts, then learns embeddings to fit this matrix.

GloVe Advantages

- Faster training (process matrix once)
- Uses global corpus statistics
- Better on word analogy tasks
- Parallelizable training

Co-occurrence Matrix

Definition

X = number of times word *j* appears in context of word *i*

Corpus: "I like deep learning. I like NLP."

X matrix (symmetric):

	I	like	deep	learning	NLP
I	0	2	0	0	0
like	2	0	1	0	1
deep	0	1	0	1	0
learning	0	0	1	0	0
NLP	0	1	0	0	0

X[I, like] = 2
X[like, deep] = 1

Matrix is symmetric: Xij = Xji | Window size affects counts

Key Formulas

Total co-occurrence for word i:

$$X_i = \sum_k X_{ik}$$

Co-occurrence probability:

$$P_{ij} = P(j|i) = X_{ij} / X_i$$

Context Window

Words within **k** positions of target word count as context (typically k = 10). Closer words often weighted more heavily.

Storage

Dense	Sparse
V × V	~1% non-zero

GloVe Objective Function

📐 GloVe Loss Function

$$J = \sum_{i,j=1}^{V} f(X_{ij})\,(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$

w_i	$\tilde{w}_j$	b_i, $\tilde{b}_j$	$f(X_{ij})$
word vec	context vec	biases	weighting

Intuition

We want the dot product of word vectors to approximate the log of co-occurrence:

$$w_i^T \tilde{w}_j \approx \log(X_{ij})$$

Similar words → similar contexts → similar embeddings

Weighting Function f(x)

Give less weight to very frequent co-occurrences (avoid domination by common words):

$$f(x) = \{\ (x/x_{max})^{\alpha} \text{ if } x < x_{max}, \text{ else } 1\ \}$$

Final embedding: **w + $\tilde{w}$** (sum of both vectors)

PART 6

Lab & Assignments

Practice & Final Project Release

Lab: N-gram Language Model

📋 Lab Tasks

Build N-gram Counts

Tokenize corpus, count unigrams, bigrams, trigrams

Implement Smoothing

Add-k smoothing, interpolation

Calculate Perplexity

Evaluate model on test set

Build Autocomplete

Predict next word, rank by probability

Key Takeaways

Language Models

N-gram models for word sequences with smoothing techniques (Add-k, Interpolation)

Word2Vec

CBOW (context→word) and Skip-gram (word→context) with Negative Sampling

Perplexity

LM evaluation metric. Lower = better model

GloVe

Count-based approach using co-occurrence matrix

Word Embeddings

Dense vectors capturing semantic similarity

Final Assignment (50%)

Choose: Chatbot, MT, Summarization, or QA. Present in Session 10.