Language Models

Lecture 6

Adarsh Kumar
Department of Industrial Mining Engineering and ICT (EMIT),
Manresa School of Engineering (EPSEM),
Polytechnic University of Catalonia, Manresa, Barcelona, Spain
adarsh.kumar@upc.edu

May 5, 2026

What is a Language Model?

Definition:
A **Language Model (LM)** is a machine learning model that predicts the next word in a sequence.

Intuition Example:

Input

"I want to drink a cup of _____"

Model Prediction

coffee (0.60), tea (0.25), water (0.10), milk (0.05)

The model predicts the most probable next word.

Why Word Prediction?

Word prediction is a crucial component of many language tasks.

- **Grammar and Spell Checking**

  Their are two midterms → There are two midterms

  Everything has improve → Everything has improved

- **Speech Recognition**

  I will be bassoon dish → I will be back soonish

Idea: The language model selects the most probable word sequence.

How Do Large Language Models (LLMs) Work?

Core Idea: LLMs are trained to predict words.

- **Training Objective**
  Learn to predict the next word given previous words:

$$P(w_t \mid w_1, w_2, ..., w_{t-1})$$

- **Left-to-Right (Autoregressive) Modeling**
  Words are predicted sequentially:

  The cat sat on the ___

  Model predicts: *mat*

- **Text Generation Process**
  1. Predict next word
  2. Append it to the sequence
  3. Repeat prediction

  Example:

  I enjoy studying $\rightarrow$ NLP $\rightarrow$ because $\rightarrow$ it $\rightarrow$ is $\rightarrow$ interesting

Text is generated by predicting the next word again and again.

Definition

1. Probability of Next Word

Given previous words $w_1, w_2, ..., w_{t-1}$:

$$P(w_t \mid w_1, w_2, ..., w_{t-1})$$

An LM assigns probabilities to **each possible next word**.

This means:

$$\sum_{w \in V} P(w \mid \text{context}) = 1$$

where V is the vocabulary.

Probability of a Whole Sentence

A language model assigns probability to an entire sentence.
For a sentence:

$$S = w_1, w_2, w_3, ..., w_n$$

Using the chain rule:

$$P(S) = \prod_{t=1}^{n} P(w_t \mid w_1, ..., w_{t-1})$$

Example:
Sentence: "I love NLP"

$$P(I) \times P(love \mid I) \times P(NLP \mid I, love)$$

How Do LLMs Generate Text?

Key Idea: LLMs predict the next word.

$$P(w_t \mid w_1, w_2, ..., w_{t-1})$$

Step-by-Step Example

Input: *"Machine learning is"*

Model predicts:

fun	0.40
powerful	0.35
difficult	0.15
bad	0.10

Chooses: **fun**

New input: *"Machine learning is fun"*

Model predicts next word again...

Generation = Predict → Append → Repeat

Two Paradigms of Language Modeling

- **N-gram Language Models** *(This lecture)*
  - Statistical models based on word counts
  - Limited context window
  - Simple and interpretable
- **Large Language Models (LLMs)** *(Neural models, future lectures)*
  - Deep neural network based
  - Capture long-range dependencies
  - Trained on massive datasets

Definition: Bag-of-Words?

Bag-of-Words (BoW) is a simple text representation technique in NLP.

- Converts text into numerical vectors
- Ignores grammar and word order
- Keeps only word frequency (counts)
- Treats document as a "bag" of words

Key Idea: Only word occurrence matters, not position.

Example

Consider two sentences:

Sentence 1

I love NLP

Sentence 2

I love machine learning

Step 1: Create Vocabulary
Combine all unique words:

$$[I,\ love,\ NLP,\ machine,\ learning]$$

This becomes the **vocabulary**.
Step 2: Word Count Representation

	I	love	NLP	machine	learning
Sentence 1	1	1	1	0	0
Sentence 2	1	1	0	1	1

Final Vector Representation
Sentence 1:

$$[1,\ 1,\ 1,\ 0,\ 0]$$

Sentence 2:

$$[1,\ 1,\ 0,\ 1,\ 1]$$

Each document is now represented as a **numerical vector**.

Why is it Called "Bag"?

- Word order is ignored
- "I love NLP"
- "NLP love I"
- Both produce the same vector

Structure is lost, frequency is preserved.

Why is BoW Important?

It was a breakthrough for converting text to numbers.

- **Feature Extraction:** Allowed classic ML algorithms (Naive Bayes, SVM) to work with text.
- **Foundation:** Built the groundwork for Topic Modeling (like LDA).
- **Simplicity:** Perfect for tasks where **keyword presence** matters more than order.

Where BoW shines (Classification Tasks):

- **Spam Detection:** "Winner," "Prize," "Free" $\rightarrow$ Spam.
- **Sentiment Analysis:** "Amazing," "Love," "Fantastic" $\rightarrow$ Positive.

BoW in Action: Sentiment Analysis

Training Data:

+ Review A: *"An amazing film with a fantastic cast."*
- Review B: *"A boring film with a terrible plot."*

Vocabulary: {amazing, film, fantastic, cast, boring, terrible, plot}

	amazing	film	fantastic	cast	boring	terrible	plot
Review A	1	1	1	1	0	0	0
Review B	0	1	0	0	1	1	1

New Review: *"The cast was amazing!"*

- Vector: [1, 0, 0, 1, 0, 0, 0]
- **Prediction:** Positive (due to "amazing" and "cast").

Note: It worked perfectly without understanding the word order!

Why BoW Fails for Text Generation

Text Generation requires a "Recipe" (Sequence). BoW is just an "Inventory List." Three Critical Failures:

1. **Loss of Word Order:**
   - "The dog bit the man" $\rightarrow$ [1,1,1,1]
   - "The man bit the dog" $\rightarrow$ [1,1,1,1] (Identical vectors, opposite meanings!)
2. **Loss of Context:**
   - "I went to the **bank** to fish." (River Bank)
   - "I went to the **bank** to deposit money." (Financial Bank)
   - BoW treats "bank" the same in both.
3. **Inability to Predict:**
   - Prompt: "My favorite color is _____"
   - BoW sees {my, favorite, color, is} (a jumbled bag). No sequence = no prediction.

The Core Problem: Order Matters

You cannot write a story with a pile of magnetic poetry words.

You need to know the sequence.

Task: Generate the next word

Input: "The cat sat on the _____"

- **Human:** Probably "mat."
- **BoW Model:** "The," "cat," "sat," "on," "the" are just unordered ingredients. It has no clue what comes next.

Verdict

BoW is **useless** for generation because generation is about **sequence**, not just **set membership**.

The Solution: Modern Language Models

To generate text, we need models that understand **context** and **sequence**.

Key Technologies:

1. **Word Embeddings:** Words are turned into rich vectors that capture meaning (not just IDs).
2. **Attention Mechanism (Transformers):** The model learns which words in the input are important to focus on for every step of generation.
3. **Positional Encoding:** The model knows the exact position of each word (Word 1, Word 2, Word 3...).

Generation Example with a Transformer:

- **Prompt:** "The capital of France is"
- **Process:** Attention links "capital" $\leftrightarrow$ "France". Internal knowledge retrieves "Paris".
- **Output:** "Paris"

Word Embeddings

Word Embeddings: The Building Blocks of LLMs

Simple Analogy

Creating a **"meaning map"** where similar words are located near each other, and relationships between words are represented as directions you can travel.

Traditional View:

- Words are discrete symbols
- "cat" $\neq$ "dog" $\neq$ "apple"
- No inherent similarity

Embedding View:

- Words are vectors
- cat $\approx$ dog (both animals)
- cat $\not\approx$ apple

Evolution: One-Hot vs Embeddings

Traditional Approach (One-Hot Encoding):

```
Vocabulary = ["cat", "dog", "apple", "orange", "run", "walk"]

cat    = [1, 0, 0, 0, 0, 0]  # 6-dimensional
dog    = [0, 1, 0, 0, 0, 0]  # All vectors are orthogonal
apple  = [0, 0, 1, 0, 0, 0]  # No similarity information
```

Problem

All words are equally different! Distance between any two words is $\sqrt{2}$.

Word Embeddings Approach:

```
cat    = [0.2, -0.4, 0.7, 0.1, -0.3, 0.5, ...]   # 300-1000 dimensions
dog    = [0.3, -0.3, 0.6, 0.2, -0.2, 0.4, ...]   # Similar to cat
apple  = [-0.1, 0.8, -0.2, 0.9, 0.1, -0.5, ...]  # Different from cat
```

Benefit

Similar words have similar vectors (short distance between them).

Key Property 1: Semantic Similarity

Words with similar meanings have similar vectors

Word	Vector (simplified)	Similarity to "happy"
"happy"	[0.8, 0.3, -0.1, 0.5]	1.00
"joyful"	[0.7, 0.4, -0.2, 0.4]	0.92
"glad"	[0.6, 0.2, -0.1, 0.3]	0.85
"sad"	[-0.6, -0.2, 0.9, -0.3]	-0.78
"angry"	[-0.5, -0.1, 0.8, -0.2]	-0.71

Semantic Clusters:

- Animals: cat, dog, tiger
- Fruits: apple, orange, banana
- Emotions: happy, sad, angry

Distance Matrix:

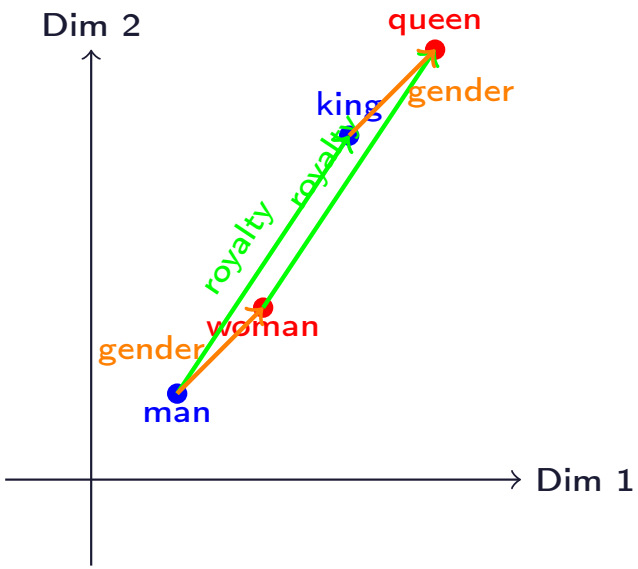
- cat-dog: 0.2 (close)
- cat-tiger: 0.3 (close)
- cat-apple: 0.9 (far)

Key Property 2: Analogical Reasoning

Embeddings capture relationships as vector arithmetic

The Famous Example

$$\text{vector("king")} - \text{vector("man")} + \text{vector("woman")} \approx \text{vector("queen")}$$



Vector relationships preserve meaning

Word Embedding Analogies

Capital Cities:	Paris − France + Italy ≈ Rome
Comparatives:	better − good + bad ≈ worse
Verb Tenses:	walking − walk + run ≈ running
Plurals:	apples − apple + cat ≈ cats
Nationalities:	Germany − Berlin + Paris ≈ France

Key Insight

The same vector offsets work across different word pairs, proving that embeddings capture **generalizable relationships**, not just memorized facts.

Relationship Type	Vector Operation
Gender	$\vec{king} - \vec{man} = \vec{queen} - \vec{woman}$
Capital	$\vec{Paris} - \vec{France} = \vec{Rome} - \vec{Italy}$
Tense	$\vec{walking} - \vec{walk} = \vec{running} - \vec{run}$

Key Property 3: Multiple Semantic Dimensions

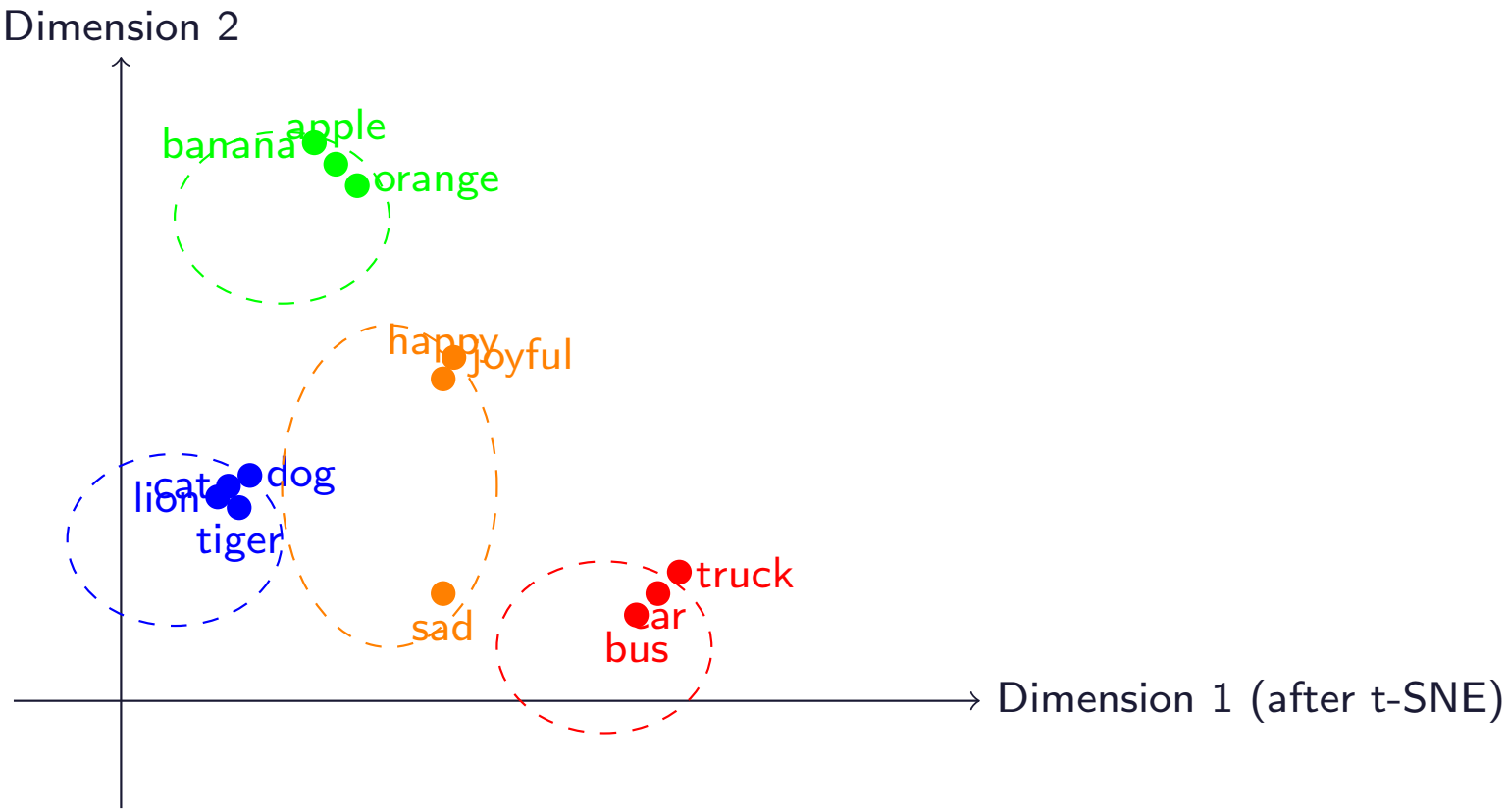
Each dimension might capture a different semantic feature

Dimension	Captures	Example
Dim 1	Gender	man (+), woman (-)
Dim 2	Formality	formal (+), casual (-)
Dim 3	Sentiment	positive (+), negative (-)
Dim 4	Abstractness	abstract (+), concrete (-)
Dim 5	Size	large (+), small (-)
Dim 6	Age	old (+), young (-)

Word Vectors (Simplified 5D)

$$\text{king} = [0.8, 0.2, 0.1, 0.9, 0.3]$$
$$\text{queen} = [0.7, 0.3, 0.2, 0.8, 0.4]$$
$$\text{man} = [0.6, 0.1, 0.3, 0.1, 0.2]$$
$$\text{woman} = [0.5, 0.2, 0.4, 0.2, 0.3]$$

Visualizing Embedding Space (2D Projection)



- Similar words form clusters
- Distance represents semantic similarity
- Different colors represent different semantic categories

How Embeddings Are Created: The Training Process

Distributional Hypothesis
Core Idea:

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

Meaning:

- Words appearing in similar contexts
- Tend to have similar meanings

Example:

- The **cat** sits on the mat
- The **dog** sits on the mat

Similar context $\Rightarrow$ Similar vectors

How Embeddings Are Created: The Training Process

Word2Vec: Skip-gram Model (predict context from target)
Goal:

$$P(\text{context word} \mid \text{target word})$$

Definition:

A *context window* is simply:

- How many words to the **left** and **right** of a target word we consider during training.

Example:

- If window size = 2, we look at:
  - 2 words to the **left** of the target word
  - 2 words to the **right** of the target word

Example Sentence:

The cat sits on the mat

Target word: **cat**
Context window (size 2):

$$\{\text{The, sits, on}\}$$

Task:

- Given "cat"
- Predict nearby words

How Embeddings Are Created: The Training Process

Neural Network Architecture:
Input $\rightarrow$ Hidden $\rightarrow$ Output

- Input: One-hot vector (size = vocabulary size)
- Hidden layer: Embedding (e.g., 300 dimensions)
- Output: Softmax probability over vocabulary

Step 1: One-Hot Encoding
Assume vocabulary size = 10,000
If "cat" is index 5321:

$$x = [0, 0, 0, \ldots, 1, \ldots, 0]$$

Properties:

- Very sparse
- High dimensional
- No semantic meaning

Step 2: Embedding Layer
We multiply:

$$x \cdot W$$

Where:

$$W \in \mathbb{R}^{10000 \times 300}$$

Since x is one-hot:

$$x \cdot W = \text{Row of } W$$

That row becomes:

$$v_{\textbf{cat}} = [0.2, -0.4, 0.7, \ldots]$$

Important: The hidden layer weights become the word embeddings.

Step 3: Predicting Context
Using embedding $v_{\textbf{cat}}$:

$$P(w \mid \text{cat}) = \text{softmax}(W' v_{\textbf{cat}})$$

Goal:

- High probability $\rightarrow$ The, sits, on
- Low probability $\rightarrow$ banana, democracy, quantum

How Learning Happens

- Compare predicted vs actual context words
- Compute loss
- Update weights using backpropagation

After training on billions of words:

$$v_{\textbf{cat}} \approx v_{\textbf{dog}}$$

Why?
Because they appear in similar contexts.
Why Analogies Work
Because relationships become directions in vector space.

$$v_{\textbf{king}} - v_{\textbf{man}} + v_{\textbf{woman}} \approx v_{\textbf{queen}}$$

The model learns:

- Gender direction
- Plural direction
- Verb tense direction

Why This Was Revolutionary
Before Word2Vec:

- Bag-of-Words
- Sparse vectors
- No semantic geometry

After Word2Vec:

- Dense vectors
- Semantic similarity
- Linear relationships

Prediction task $\Rightarrow$ Semantic structure emerges

Word2Vec and Related Models Comparison

Model	Architecture	Training Speed	Quality	Best For
Skip-gram	Target → Context	Slow	High	Rare words
CBOW	Context → Target	Fast	Medium	Frequent words
SGNS	Skip-gram + Neg Sampling	Medium	Very High	General purpose
Hierarchical Softmax	Tree-based	Very Fast	Medium	Large vocabularies
FastText	Subword + Skip-gram	Medium	High	OOV words, morphology
GloVe	Co-occurrence matrix	Fast	High	Word analogies
Doc2Vec	Document + Words	Slow	High	Document similarity

Table: Comparison of Word2Vec models and related embedding techniques

Note: SGNS = Skip-gram with Negative Sampling, OOV = Out-of-Vocabulary

Word2Vec Continuous Bag of Words (CBOW): Overview

Core Idea: Predict a target word from its surrounding context words

Analogy: Fill in the Blank

"The _____ sat on the mat" $\rightarrow$ Model predicts: "cat"

Key Properties:

- Continuous vector representations
- Bag of words (order doesn't matter)
- Context window of size m

Architecture:

- Input: Context words
- Hidden: Average of embeddings
- Output: Target word probability

Problem Formulation

Given: Sequence of words $w_1, w_2, ..., w_T$

Context Window

For target word w_t, context of size m:

$$\text{Context}(w_t) = \{w_{t-m}, ..., w_{t-1}, w_{t+1}, ..., w_{t+m}\}$$

Numerical Example

Sentence: "The cat sat on the mat"

Target	Position	Context (m=2)
"cat"	$t = 2$	{The, sat, on, the}
"sat"	$t = 3$	{cat, on, the, mat}
"on"	$t = 4$	{sat, the, mat}

Objective: Maximize $P(w_t|\text{Context}(w_t))$

Mathematical Notation

Symbol	Meaning
V	Vocabulary size
N	Embedding dimension
$W \in^{V \times N}$	Input weight matrix
$W' \in^{N \times V}$	Output weight matrix
$x_k \in^{V}$	One-hot vector for word k
$v_w \in^{N}$	Input embedding of word w
$v'_w \in^{N}$	Output vector of word w
C	Number of context words
η	Learning rate

Numerical Values

$V = 6$ words

$N = 3$ (simplified)

$C = 4$ (for "cat")

Vocabulary: {The, cat, sat, on, the, mat}

Word Indices: The:0, cat:1, sat:2, on:3, the:4, mat:5

Initial Weight Matrices

Input Weight Matrix W (Random Initialization)

$$W = \begin{pmatrix} 0.1 & -0.2 & 0.3 \\ -0.1 & 0.4 & 0.2 \\ 0.3 & 0.1 & -0.3 \\ -0.2 & 0.2 & 0.1 \\ 0.4 & -0.1 & 0.2 \\ 0.2 & 0.3 & -0.1 \end{pmatrix} \in^{6\times 3}$$

Output Weight Matrix W' (Random Initialization)

$$W' = \begin{pmatrix} 0.2 & 0.1 & -0.2 & 0.3 & -0.1 & 0.2 \\ -0.3 & 0.2 & 0.1 & -0.2 & 0.4 & 0.1 \\ 0.1 & -0.2 & 0.3 & 0.1 & -0.3 & 0.4 \end{pmatrix} \in^{3\times 6}$$

Note: Each row of W is the embedding v_w, each column of W' is v'_w

Forward Pass Step 1: One-Hot Encoding

Training Example: Target "cat" with context ["The", "sat", "on", "the"]

One-Hot Vectors

$$
\begin{aligned}
x_{\text{The}} &= [1, 0, 0, 0, 0, 0] \\
x_{\text{sat}} &= [0, 0, 1, 0, 0, 0] \\
x_{\text{on}} &= [0, 0, 0, 1, 0, 0] \\
x_{\text{the}} &= [0, 0, 0, 0, 1, 0] \\
x_{\text{target}} &= [0, 1, 0, 0, 0, 0] \quad \text{(for "cat")}
\end{aligned}
$$

Word Index Mapping

Word	Index
"The"	0
"cat"	1
"sat"	2
"on"	3
"the"	4
"mat"	5

Forward Pass Step 2: Get Word Embeddings

Embedding Lookup: $v_w = W^T x_w$ (or simply the corresponding row of W)

$$
\begin{aligned}
v_{\text{The}} &= [0.1, -0.2, 0.3] \quad \text{(row 0)} \\
v_{\text{sat}} &= [0.3, 0.1, -0.3] \quad \text{(row 2)} \\
v_{\text{on}} &= [-0.2, 0.2, 0.1] \quad \text{(row 3)} \\
v_{\text{the}} &= [0.4, -0.1, 0.2] \quad \text{(row 4)}
\end{aligned}
$$

Verification

For $x_{\text{The}} = [1, 0, 0, 0, 0, 0]$:

$$
W^T x_{\text{The}} = \begin{pmatrix} 0.1 & -0.1 & 0.3 & -0.2 & 0.4 & 0.2 \\ -0.2 & 0.4 & 0.1 & 0.2 & -0.1 & 0.3 \\ 0.3 & 0.2 & -0.3 & 0.1 & 0.2 & -0.1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.1 \\ -0.2 \\ 0.3 \end{pmatrix}
$$

Forward Pass Step 3: Average Context Vectors

Hidden Layer: Average of all context word embeddings

$$h = \frac{1}{C} \sum_{i=1}^{C} v_{w_i}$$

Numerical Calculation

$$\begin{aligned} h &= \frac{1}{4}(v_{\text{The}} + v_{\text{sat}} + v_{\text{on}} + v_{\text{the}}) \\ &= \frac{1}{4}([0.1, -0.2, 0.3] + [0.3, 0.1, -0.3] + [-0.2, 0.2, 0.1] + [0.4, -0.1, 0.2]) \\ &= \frac{1}{4}([0.6, 0.0, 0.3]) \\ &= [0.15, 0.0, 0.075] \end{aligned}$$

Interpretation

$h \in^3$ is the distributed representation of the context "The sat on the"

Forward Pass Step 4: Score Computation

Output Scores: $u_j = v'^T_j h$ for each word j

$$u = W'^T h \in^V$$

Numerical Calculation

$$u = \begin{pmatrix} 0.2 & -0.3 & 0.1 \\ 0.1 & 0.2 & -0.2 \\ -0.2 & 0.1 & 0.3 \\ 0.3 & -0.2 & 0.1 \\ -0.1 & 0.4 & -0.3 \\ 0.2 & 0.1 & 0.4 \end{pmatrix}^T \begin{pmatrix} 0.15 \\ 0.0 \\ 0.075 \end{pmatrix}$$

Compute each u_j:

$$u_0 = 0.2(0.15) + (-0.3)(0) + 0.1(0.075) = 0.03 + 0 + 0.0075 = 0.0375$$
$$u_1 = 0.1(0.15) + 0.2(0) + (-0.2)(0.075) = 0.015 + 0 - 0.015 = 0.0$$
$$u_2 = (-0.2)(0.15) + 0.1(0) + 0.3(0.075) = -0.03 + 0 + 0.0225 = -0.0075$$
$$u_3 = 0.3(0.15) + (-0.2)(0) + 0.1(0.075) = 0.045 + 0 + 0.0075 = 0.0525$$
$$u_4 = (-0.1)(0.15) + 0.4(0) + (-0.3)(0.075) = -0.015 + 0 - 0.0225 = -0.0375$$
$$u_5 = 0.2(0.15) + 0.1(0) + 0.4(0.075) = 0.03 + 0 + 0.03 = 0.06$$

Forward Pass Step 5: Softmax Probability

Softmax Function: $y_j = \frac{\exp(u_j)}{\sum_{k=1}^{V} \exp(u_k)}$

Numerical Calculation

First compute $\exp(u_j)$:

$$\exp(u_0) = \exp(0.0375) = 1.0382$$
$$\exp(u_1) = \exp(0.0) = 1.0000$$
$$\exp(u_2) = \exp(-0.0075) = 0.9925$$
$$\exp(u_3) = \exp(0.0525) = 1.0539$$
$$\exp(u_4) = \exp(-0.0375) = 0.9632$$
$$\exp(u_5) = \exp(0.06) = 1.0618$$

Sum $= 1.0382 + 1.0000 + 0.9925 + 1.0539 + 0.9632 + 1.0618 = 6.1096$

$$y_0 = 1.0382/6.1096 = 0.1699$$
$$y_1 = 1.0000/6.1096 = 0.1637 \quad \text{(target "cat")}$$
$$y_2 = 0.9925/6.1096 = 0.1625$$
$$y_3 = 1.0539/6.1096 = 0.1725$$
$$y_4 = 0.9632/6.1096 = 0.1576$$
$$y_5 = 1.0618/6.1096 = 0.1738$$

Loss Function: Negative Log-Likelihood

Loss for a single example:

$$= -\log P(w_t|\text{context}) = -\log y_t$$

Numerical Calculation

Target word is "cat" (index 1), so $y_t = y_1 = 0.1637$:

$$= -\log(0.1637) = -(-1.8097) = 1.8097$$

$$= -u_t + \log \sum_{k=1}^{V} \exp(u_k)$$
$$= -0.0 + \log(6.1096)$$
$$= 0 + 1.8097 = 1.8097 \quad \checkmark$$

Total loss over corpus:

$$_{\text{total}} = -\sum_{t=1}^{T} \log P(w_t|\text{context}_t)$$

Backpropagation Step 1: Output Layer Gradients

Prediction Error: $e_j = y_j - \delta_{jt}$ where δ_{jt} is 1 if $j = t$, else 0

Numerical Error Calculation

Target $t = 1$ ("cat"):

$$
\begin{aligned}
e_0 &= y_0 - 0 = 0.1699 \\
e_1 &= y_1 - 1 = 0.1637 - 1 = -0.8363 \\
e_2 &= y_2 - 0 = 0.1625 \\
e_3 &= y_3 - 0 = 0.1725 \\
e_4 &= y_4 - 0 = 0.1576 \\
e_5 &= y_5 - 0 = 0.1738
\end{aligned}
$$

Gradient for output vectors:

$$\frac{\partial}{\partial v'_j} = e_j \cdot h$$

Example for $j = 1$ (target)

$$\frac{\partial}{\partial v'_1} = (-0.8363) \cdot [0.15, 0.0, 0.075] = [-0.1254, 0.0, -0.0627]$$

Backpropagation Step 2: Update Output Vectors

Update Rule: $v_j'^{(\text{new})} = v_j'^{(\text{old})} - \eta \frac{\partial}{\partial v_j'}$

Numerical Update ($\eta = 0.1$)

Original $v_1' = \begin{pmatrix} 0.1 \\ 0.2 \\ -0.2 \end{pmatrix}$ (column 1 of W')

Gradient $\frac{\partial}{\partial v_1'} = \begin{pmatrix} -0.1254 \\ 0.0 \\ -0.0627 \end{pmatrix}$

Update:

$$v_1'^{(\text{new})} = \begin{pmatrix} 0.1 \\ 0.2 \\ -0.2 \end{pmatrix} - 0.1 \cdot \begin{pmatrix} -0.1254 \\ 0.0 \\ -0.0627 \end{pmatrix}$$

$$v_1'^{(\text{new})} = \begin{pmatrix} 0.1 + 0.01254 \\ 0.2 - 0 \\ -0.2 + 0.00627 \end{pmatrix} = \begin{pmatrix} 0.11254 \\ 0.2 \\ -0.19373 \end{pmatrix}$$

Similar updates for all v_j' (all 6 output vectors)

Backpropagation Step 3: Hidden Layer Gradient

Gradient with respect to hidden layer:

$$\frac{\partial}{\partial h} = \sum_{j=1}^{V} e_j \cdot v'_j$$

Numerical Calculation

Using all e_j and original v'_j:

$$\frac{\partial}{\partial h} = 0.1699 \begin{pmatrix} 0.2 \\ -0.3 \\ 0.1 \end{pmatrix} + (-0.8363) \begin{pmatrix} 0.1 \\ 0.2 \\ -0.2 \end{pmatrix} + 0.1625 \begin{pmatrix} -0.2 \\ 0.1 \\ 0.3 \end{pmatrix} + 0.1725 \begin{pmatrix} 0.3 \\ -0.2 \\ 0.1 \end{pmatrix} + 0.1576 \begin{pmatrix} -0.1 \\ 0.4 \\ -0.3 \end{pmatrix} + 0.1738 \begin{pmatrix} 0.2 \\ 0.1 \\ 0.4 \end{pmatrix}$$

Compute component-wise:

$$\frac{\partial}{\partial h_1} = 0.1699(0.2) - 0.8363(0.1) + 0.1625(-0.2) + 0.1725(0.3) + 0.1576(-0.1) + 0.1738(0.2)$$

$$= 0.03398 - 0.08363 - 0.0325 + 0.05175 - 0.01576 + 0.03476 = -0.0114$$

Continue for all components: $\frac{\partial}{\partial h} = [-0.0114, 0.0852, -0.0231]$