

WORD EMBEDDING

(word2vec)

MATHS FOUNDATION
for
COMPUTER SCIENCE (CO5097)

Ly Minh Trung
Ngo Le Khoa
Bui Minh Hieu
Tran Duy Quang
Nguyen Anh Tai

- **Introduction**
- **Skip-gram**
- **Continuous Bag of words (CBOW)**
- **Summary**

1. Introduction

- Problem statement:

The Problem: One-Hot Encoding

- **Representation:** Sparse vectors with size = Vocabulary (V).
- **Curse of Dimensionality:** Computationally expensive and wasteful for large vocabularies.
- **Lack of Semantics:** Vectors are orthogonal (dot product = 0). The model cannot detect relationships (e.g., "Man" vs. "Woman").

The Solution: Word Embedding (Word2Vec)

- **Representation:** Dense vectors in lower dimensions (e.g., 100–300).
- **Semantic Awareness:** Captures meaning. Semantically similar words are geometrically close.

1. Introduction

- Applications:

Key Applications

- **Machine Translation:** Maps cross-lingual vocabulary similarities.
- **Sentiment Analysis:** Recognizes context (e.g., "Great" ~ "Excellent").
- **Information Retrieval:** Semantic search beyond exact keyword matching.
- **Vector Arithmetic:** Solves analogies (e.g., King - Man + Woman ~ Queen).

The Fundamental Shift

- **Unstructured -> Structured:** Converts raw text into numerical vectors.
- **Enables Deep Learning:** Allows optimization via **Gradient Descent**.
- **Geometry:** Transforms linguistic problems into spatial geometry.

2. Skip-gram

- Model Architecture:

Skip-gram Overview

- **Core Concept:**
 1. The Skip-gram model predicts **context words** (surrounding words) given a **center word**.
 2. It is the inverse of the CBOW model.
- **Goal:**
 1. To learn high-quality vector representations (embeddings) where semantically similar words are close in the vector space.
- **Architecture Flow:**
 1. **Input:** One-hot vector of the center word (w_{in}).
 2. **Projection:** Multiplication with the weight matrix (W_{in}).
 3. **Output:** Probability distribution of context words (w_{t-m}, \dots, w_{t+m}).

2. Skip-gram

- Model Architecture:

Formal Definitions

Vector Representations:

- For every word i in vocabulary V , we define two vectors:
 - $\mathbf{v}_i \in \mathbb{R}^d$: Vector when word i is the **center** word.
 - $\mathbf{u}_i \in \mathbb{R}^d$: Vector when word i is the **context** word.

Softmax Function:

- Defines the probability of observing context word w_o given center word w_c :

$$P(w_o|w_c) = \frac{\exp(\mathbf{u}_o^\top \mathbf{v}_c)}{\sum_{j=1}^{|V|} \exp(\mathbf{u}_j^\top \mathbf{v}_c)}$$

2. Skip-gram

- Model Architecture:

Likelihood:

- Maximize the probability of observing valid context words across the entire corpus of length T with window size m :

$$L = \prod_{t=1}^T \prod_{-m \leq j \leq m, j \neq 0} P(w^{(t+j)} | w^{(t)})$$

Loss Function (Negative Log-Likelihood):

- For optimization (Gradient Descent), we minimize the negative log-likelihood:

$$J = - \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log P(w^{(t+j)} | w^{(t)})$$

2. Skip-gram

- Detail Computation:

Forward Pass

- **Step 1: Lookup**
 - Retrieve the vector representation (\mathbf{v}_c) for the current center word from the input matrix.
- **Step 2: Score Calculation (Dot Product)**
 - Calculate the similarity score (z) between the center vector and all context vectors:

$$z_i = \mathbf{u}_i^\top \cdot \mathbf{v}_c$$

- **Step 3: Probability Distribution**
 - Apply the **Softmax** function to convert scores (z) into probabilities (\hat{y}) summing to 1.

2. Skip-gram

- Detail Computation:

Loss Calculation (Example)

Scenario:

- Sentence: *"king is a man"*
- Center Word: **"is"** → Target Context: **"king"**

Computation:

- The model computes the probability: $P(\text{"king"} \mid \text{"is"}) \approx 0.077\%$
- Calculate Loss:

$$\text{Loss} = -\log(0.077) \approx 2.56$$

Insight:

- The probability is relatively low (~7.7%) because "is" is a common **stop-word** connected to many nouns (king, queen, prince, etc.), diluting the probability mass.

2. Skip-gram

- Detail Computation:

Backward Pass (Optimization)

Gradient Calculation:

- Compute the error between the predicted probability (\hat{y}) and the actual label (y):

$$\text{Error} = \hat{y} - y$$

Stochastic Gradient Descent (SGD):

- Update the vector representations to minimize error:

$$\mathbf{v}_{new} = \mathbf{v}_{old} - \eta \cdot \text{Error} \cdot \mathbf{u}_{context}$$

- *Intuition:* If the prediction is wrong, "push" the vectors closer or further apart.

2. Skip-gram

- Corresponding Implementation in Python:

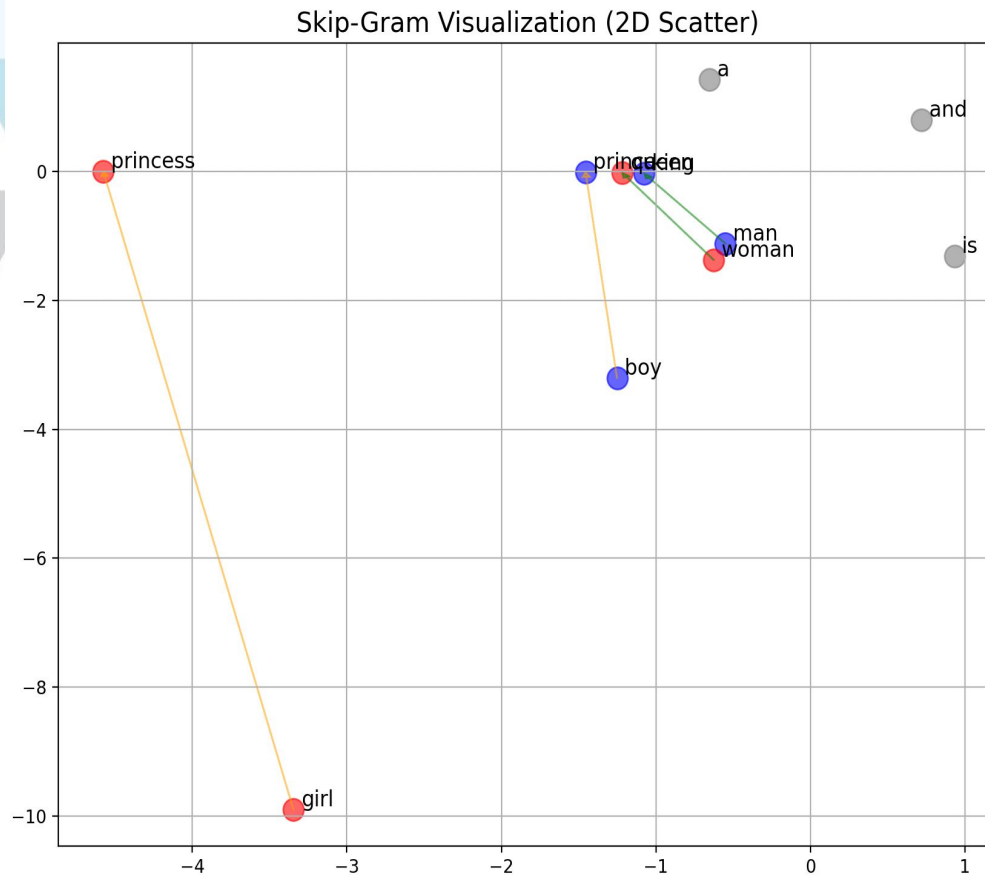
Implementation Logic (Manual SGD)

```
# =====  
# 2. TRAINING SKIP-GRAM (MANUAL)  
# =====  
print("\n--- TRAINING SKIP-GRAM (MANUAL) ---")  
  
W1_sg = torch.randn(V, d, requires_grad=True) # Center Vectors  
W2_sg = torch.randn(d, V, requires_grad=True) # Context Vectors  
  
for epoch in range(epochs):  
    total_loss = 0  
    for target_idx, context_idx in skipgram_data:  
        # Forward  
        h = W1_sg[target_idx]                # Lookup center vector  
        z = torch.matmul(h, W2_sg)           # Score calculation  
        y_hat = F.softmax(z, dim=0)          # Softmax  
  
        # Loss  
        loss = -torch.log(y_hat[context_idx])  
        total_loss += loss.item()  
  
        # Backward  
        loss.backward()  
  
        # Update (SGD)  
        with torch.no_grad():  
            W1_sg -= lr * W1_sg.grad  
            W2_sg -= lr * W2_sg.grad  
            W1_sg.grad.zero_()  
            W2_sg.grad.zero_()  
  
    if (epoch+1) % 500 == 0:  
        print(f"Epoch {epoch+1}: Loss = {total_loss:.4f}")  
  
sg_embeddings = W1_sg.detach().numpy()
```

2. Skip-gram

- Corresponding Implementation in Python:

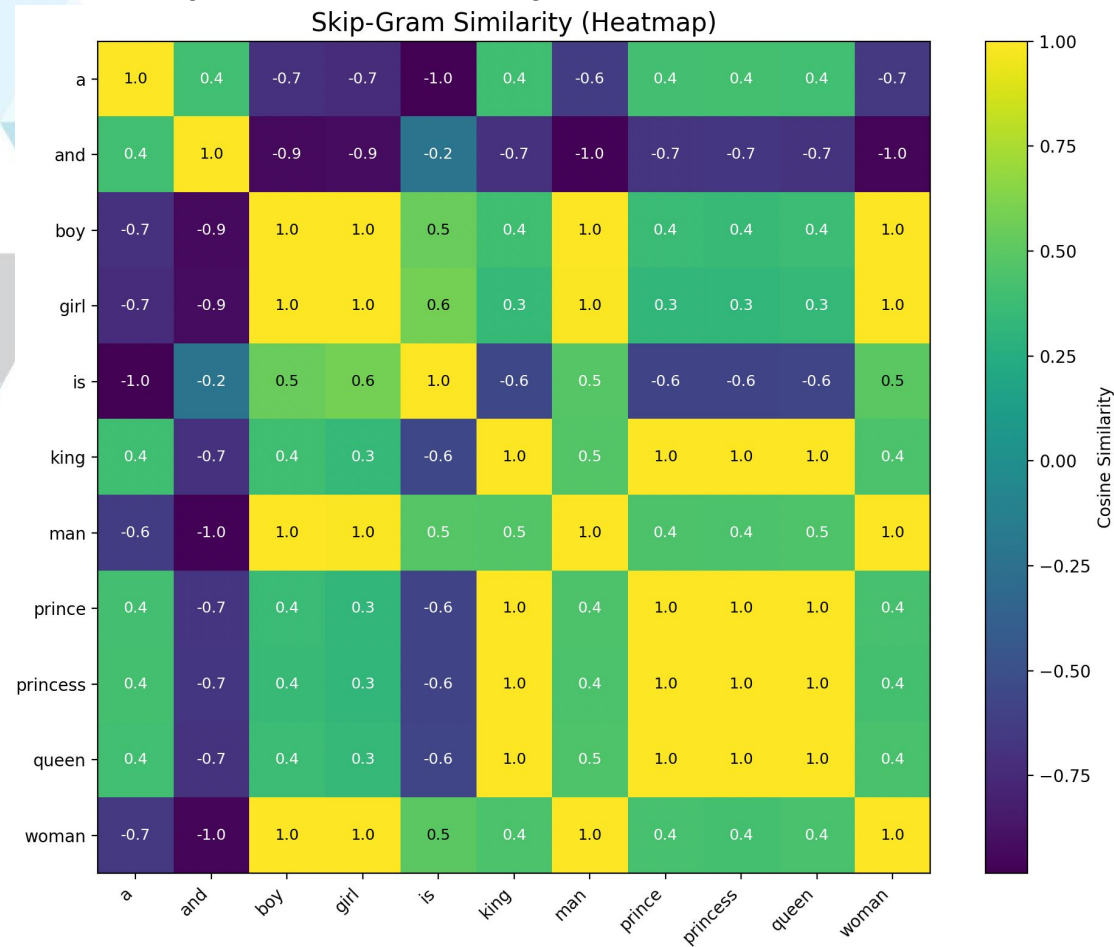
Visualization 1 - 2D Scatter Plot



2. Skip-gram

- Corresponding Implementation in Python:

Visualization 2 - Heatmap (Cosine Similarity)



3. CBOW

- Model Architecture:

CBOW Overview

- **Core Concept:**
 1. The CBOW model predicts the **center word** (target) given **context words** (surrounding words).
 2. It is the inverse of the Skip-gram model.
- **Goal:**
 1. To learn high-quality vector representations (embeddings) by aggregating context information to predict the target.
- **Architecture Flow:**
 1. **Input:** One-hot vectors of context words (w_{t-m}, \dots, w_{t+m}).
 2. **Projection:** Multiplication with weight matrix (W_{in}) and **averaging** the resulting vectors.
 3. **Output:** Probability distribution of the center word (w_c) or (w_t).

3. CBOW

- Model Architecture:

Formal Definitions

Vector Representations:

- $\mathbf{v}_i \in \mathbb{R}^d$: vector when the word serves as a **context** word.
- $\mathbf{u}_i \in \mathbb{R}^d$: vector when the word serves as a **center** (target) word.

Softmax:

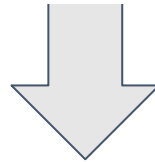
$$P(w_c | \mathcal{W}_o) = \frac{\exp(\mathbf{u}_c^\top \bar{\mathbf{v}}_o)}{\sum_{i \in V} \exp(\mathbf{u}_i^\top \bar{\mathbf{v}}_o)}$$

3. CBOW

- Model Architecture:

$\bar{\mathbf{v}}_o$ be their mean vector:

$$\bar{\mathbf{v}}_o = \frac{1}{2m} \sum_{i=1}^{2m} \mathbf{v}_{o_i}$$



$$P(w_c \mid w_{o_1}, \dots, w_{o_{2m}}) = \frac{\exp \left(\frac{1}{2m} \mathbf{u}_c^\top (\mathbf{v}_{o_1} + \dots + \mathbf{v}_{o_{2m}}) \right)}{\sum_{i \in V} \exp \left(\frac{1}{2m} \mathbf{u}_i^\top (\mathbf{v}_{o_1} + \dots + \mathbf{v}_{o_{2m}}) \right)}$$

3. CBOW

- Model Architecture:

Likelihood: Maximize the probability of the correct center word across the entire corpus sequence of length T .

$$\prod_{t=1}^T P(w^{(t)} \mid w^{(t-m)}, \dots, w^{(t-1)}, w^{(t+1)}, \dots, w^{(t+m)})$$

Maximizing likelihood is mathematically equivalent to **minimizing the Global Negative Log-Likelihood (Loss Function)**

$$J = - \sum_{t=1}^T \log P(w^{(t)} \mid w^{(t-m)}, \dots, w^{(t-1)}, w^{(t+1)}, \dots, w^{(t+m)})$$

3. CBOW

- Model Architecture:

For a target word w_c and context \mathcal{W}_o , the log probability is defined as:

$$\log P(w_c | \mathcal{W}_o) = \mathbf{u}_c^\top \bar{\mathbf{v}}_o - \log \left(\sum_{i \in V} \exp(\mathbf{u}_i^\top \bar{\mathbf{v}}_o) \right)$$

3. CBOW

- Model Architecture:

Optimization Strategy

- **Method:** Weights are optimized using **Stochastic Gradient Descent (SGD)**.
- **Process:** Gradients are computed by differentiating the log-likelihood loss function with respect to specific context word vectors \mathbf{v}_{o_i} (where $i = 1, \dots, 2m$).

Gradient Update Rule

- The gradient describes the "error" between the actual target vector and the model's probabilistic expectation:

$$\frac{\partial J}{\partial \mathbf{v}_{o_i}} = \frac{1}{2m} \left(\sum_{j \in V} P(w_j | \mathcal{W}_o) \mathbf{u}_j - \mathbf{u}_c \right)$$

$$\frac{\partial \log P(w_c | \mathcal{W}_o)}{\partial \mathbf{v}_{o_i}} = \frac{1}{2m} \left(\mathbf{u}_c - \sum_{j \in V} \frac{\exp(\mathbf{u}_j^\top \bar{\mathbf{v}}_o)}{\sum_{i \in V} \exp(\mathbf{u}_i^\top \bar{\mathbf{v}}_o)} \mathbf{u}_j \right)$$

3. CBOW

- Model Architecture:

$$J = -\log P(w_c | \mathcal{W}_o) = - \left(\mathbf{u}_c^\top \bar{\mathbf{v}}_o - \log \sum_{j \in V} \exp(\mathbf{u}_j^\top \bar{\mathbf{v}}_o) \right) \longrightarrow J = \underbrace{-\mathbf{u}_c^\top \bar{\mathbf{v}}_o}_{\text{Phần 1}} + \underbrace{\log \left(\sum_{j \in V} \exp(\mathbf{u}_j^\top \bar{\mathbf{v}}_o) \right)}_{\text{Phần 2}}$$

- $f(x) = -ax \rightarrow f'(x) = -a$

$$\frac{\partial}{\partial \bar{\mathbf{v}}_o} (-\mathbf{u}_c^\top \bar{\mathbf{v}}_o) = -\mathbf{u}_c$$

- $\ln(u)' = u' / u$

$$\frac{\partial}{\partial \bar{\mathbf{v}}_o} \log \left(\sum_{j \in V} \exp(\mathbf{u}_j^\top \bar{\mathbf{v}}_o) \right) = \frac{1}{\sum_{k \in V} \exp(\mathbf{u}_k^\top \bar{\mathbf{v}}_o)} \cdot \frac{\partial}{\partial \bar{\mathbf{v}}_o} \left(\sum_{j \in V} \exp(\mathbf{u}_j^\top \bar{\mathbf{v}}_o) \right)$$

3. CBOW

- Model Architecture:

- $[\mathbf{e}^{(u)}]' = \mathbf{u}' \cdot \mathbf{e}^u$

$$= \frac{1}{\sum \exp(\dots)} \cdot \sum_{j \in V} \exp(\mathbf{u}_j^\top \bar{\mathbf{v}}_o) \mathbf{u}_j$$

3. CBOW

- Model Architecture:

General Weight Update Formula for context words's vectors

$$\mathbf{v}_{o_i}^{(new)} = \mathbf{v}_{o_i}^{(old)} - \eta \cdot \left(\frac{1}{2m} \sum_{w \in V} (\hat{y}_w - y_w) \mathbf{u}_w \right)$$

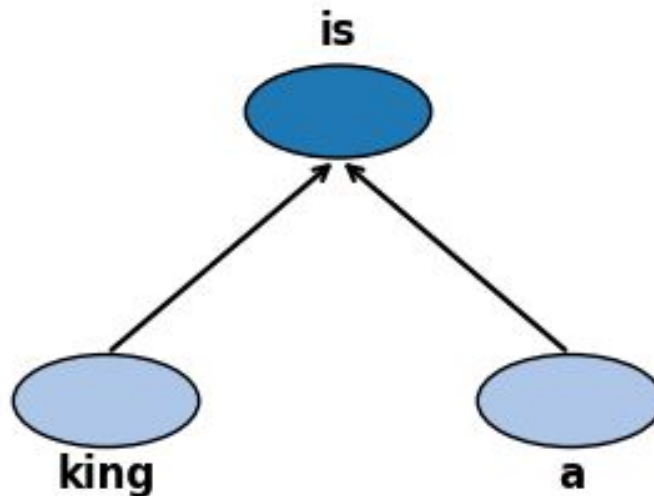
General Weight Update Formula for target word's vector

$$\mathbf{u}_w^{(new)} = \mathbf{u}_w^{(old)} - \eta \cdot \left((\hat{y}_w - y_w) \cdot \mathbf{h} \right)$$

3. CBOW

- Detail Computation:

Scenario: We take the sentence: "**king** is a man".



3. CBOW

- Detail Computation:

Step 1: Initialization & Matrix Lookup

1. Input Matrix (W_{in})

Index	Word	Char 1 (column 0)	Char 2 (column 1)	Remark
0	king	0.2	0.9	this is v_{king}
1	is	-0.5	0.1	
2	a	0.1	0.3	this is v_a
3	man	0.8	-0.2	
...	

3. CBOW

- Detail Computation:

Step 1: Initialization & Matrix Lookup

2. Output Matrix (W_{out})

	king (0)	is (1)	a (2)	man (3)	...
Char 1	0.01	0.5	0.1	0.1	...
Char 2	0.99	1.0	0.8	0.4	...
Vector	u_{king}	u_{is}	u_a	u_{man}	

3. CBOW

- Detail Computation:

Step 2: Hidden Layer Projection

We average the input vectors of the context words to form the hidden layer vector h (the aggregate meaning of the context)

$$h = \frac{1}{2m}(v_{king} + v_a)$$

$$h = \frac{1}{2} ([0.20, 0.90] + [0.10, 0.30])$$

$$h = \frac{1}{2} [0.30, 1.20]$$

$$h = [0.15, 0.60]$$

3. CBOW

- Detail Computation:

Step 3: Score Calculation (Logits)

Taking the dot product of the hidden vector h with the output vectors of all words in the vocabulary (*Note: In a full implementation, this is calculated for all words in V*).

$$z_{is} = uT_{is} \cdot h = (0.50 \times 0.15) + (1.00 \times 0.60)$$

$$z_{is} = 0.075 + 0.60 = 0.675$$

$$z_{man} = uT_{man} \cdot h = (0.10 \times 0.15) + (0.40 \times 0.60)$$

$$z_{man} = 0.015 + 0.24 = 0.255$$

3. CBOW

- Detail Computation:

Step 4: Probability Estimation (Softmax)

Assuming for this simplified example that our vocabulary only consists of these two words ("is", "man"), but in real case the vocabulary must contain all words in the input corpus:

Exponentiate the scores:

$$e^{z_{is}} = e^{0.675} \approx 1.964$$

$$e^{z_{man}} = e^{0.255} \approx 1.290$$



Calculate the sum:

$$\Sigma = 1.964 + 1.290 = 3.254$$

Calculate Probabilities:

$$P(\text{"is"} \mid \text{context}) = \frac{1.964}{3.254} \approx 0.6035 \text{ (60.35\%)}$$

$$P(\text{"man"} \mid \text{context}) = \frac{1.290}{3.254} \approx 0.3965 \text{ (39.65\%)}$$

3. CBOW

- Detail Computation:

Step 5: Loss Calculation

the Negative Log-Likelihood

$$\text{Loss} = -\log(P(\text{"is"} \mid \text{context}))$$

$$\text{Loss} = -\log(0.6035) \approx 0.505$$

3. CBOW

- Detail Computation:

Step 6: Backpropagation and Weight Update

Calculate Error Signal (e): The error is the difference between the predicted probability (\hat{y}) and the actual target (y).

$$\text{Error} = \hat{y} - y$$

Target “is” ($y = 1$): $e_{\text{is}} = 0.6035 - 1 = -0.3965$

Non-target “man” ($y=0$) $e_{\text{man}} = 0.3965 - 0 = 0.3965$

3. CBOW

- Detail Computation:

Step 6: Backpropagation and Weight Update

Calculate Gradient w.r.t Hidden Layer (h): backtrack the error through the Output Matrix (W_{out}). The gradient for h is the weighted sum of errors multiplied by their respective target vectors.

$$\frac{\partial Loss}{\partial h} = (e_{is} \cdot u_{is}) + (e_{man} \cdot u_{man})$$



	king (0)	is (1)	a (2)	man (3)	...
Char 1	0.01	0.5	0.1	0.1	...
Char 2	0.99	1.0	0.8	0.4	...
Vector	u_{king}	u_{is}	u_a	u_{man}	

$$\frac{\partial J}{\partial h} = \sum_{w \in V} (\hat{y}_w - y_w) \mathbf{u}_w = \underbrace{(\hat{y}_{is} - 1) \cdot \mathbf{u}_{is}} + \underbrace{(\hat{y}_{king} - 0) \cdot \mathbf{u}_{king}} + \underbrace{(\hat{y}_a - 0) \cdot \mathbf{u}_a} + \underbrace{(\hat{y}_{man} - 0)}$$

Contribution from "is": $-0.3965 \times [0.50, 1.00] = [-0.198, -0.396]$

Contribution from "man": $0.3965 \times [0.10, 0.40] = [0.040, 0.159]$

Total Gradient: $[-0.198 + 0.040, -0.396 + 0.159] = [-0.158, -0.237]$

3. CBOW

- Detail Computation:

Step 6: Backpropagation and Weight Update

Calculate Gradient w.r.t Input Vector (v_{king}): The gradient passes through the averaging operation. Since we used 2 context words ($2m=2$), the gradient is distributed equally:

$$\frac{\partial \text{Loss}}{\partial v_{\text{king}}} = \frac{\partial \text{Loss}}{\partial h} \times \frac{1}{2}$$

$$\text{Grad.} v_{\text{king}} = \frac{1}{2} \times [-0.158, -0.237] = [-0.079, -0.118]$$

3. CBOW

- Detail Computation:

Step 6: Backpropagation and Weight Update

Update the Vector using Learning Rate (η): We apply the Gradient Descent update rule. Let's assume a learning rate $\eta = 0.1$. Let take “king” an example:

$$V_{\text{king}}^{(\text{new})} = V_{\text{king}}^{(\text{old})} - \eta \cdot \text{Grad}.V_{\text{king}}$$

- With $V_{\text{king}}^{(\text{old})} = [0.20, 0.90]$

$$V_{\text{king}}^{(\text{new})} = [0.20, 0.90] - 0.1 \times [-0.079, -0.118] = [-0.0079, -0.0118]$$

$$= [0.20, 0.90] - [-0.0079, -0.0118]$$

$$= [0.20 + 0.0079, 0.90 + 0.0118]$$

$$\approx [0.208, 0.912]$$

3. CBOW

- Corresponding Implementation in Python:

Implementation Logic:

```
print("\n--- TRAINING CBOW (MANUAL) ---")

# 1. Khởi tạo lại trọng số mới
W1_cbow = torch.randn(V, d, requires_grad=True) # Context Matrix (Input)
W2_cbow = torch.randn(d, V, requires_grad=True) # Center Matrix (Output)

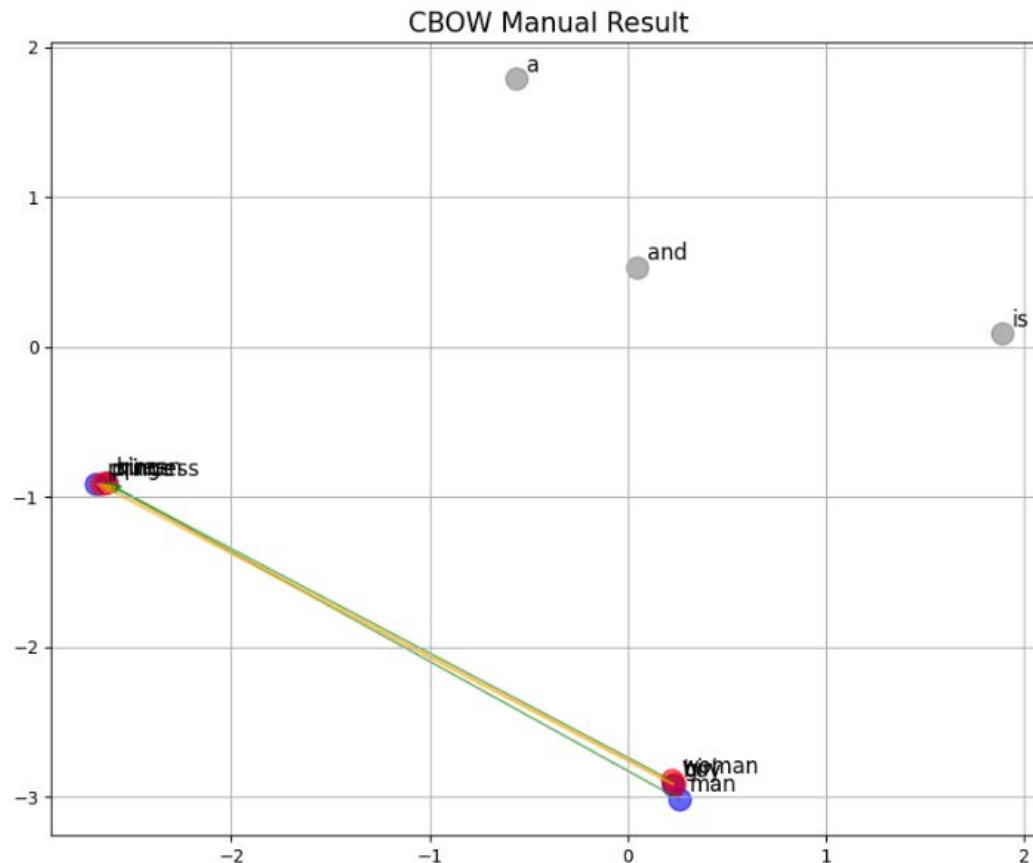
for epoch in range(epochs):
    total_loss = 0
    for context_idxs, target_idx in cbow_data:
        # --- FORWARD PASS (TÍNH TAY) ---
        # Bước 1: Lấy các vector context
        context_vecs = W1_cbow[context_idxs]
        # Bước 2: TÍNH TRUNG BÌNH (Đặc trưng của CBOW)
        h = torch.mean(context_vecs, dim=0) # Shape: [d]
        # Bước 3: Tính Score (Dot product với output matrix)
        z = torch.matmul(h, W2_cbow) # Shape: [V]
        # Bước 4: Softmax
        y_hat = F.softmax(z, dim=0)
        # --- LOSS ---
        loss = -torch.log(y_hat[target_idx])
        total_loss += loss.item()
        # --- BACKWARD ---
        loss.backward()
        # --- UPDATE ---
        with torch.no_grad():
            W1_cbow -= lr * W1_cbow.grad
            W2_cbow -= lr * W2_cbow.grad
            W1_cbow.grad.zero_()
            W2_cbow.grad.zero_()
    if (epoch+1) % 500 == 0:
        print(f"Epoch {epoch+1}: Loss = {total_loss:.4f}")

# Lưu kết quả
cbow_embeddings = W1_cbow.detach().numpy()
```

3. CBOW

- Corresponding Implementation in Python:

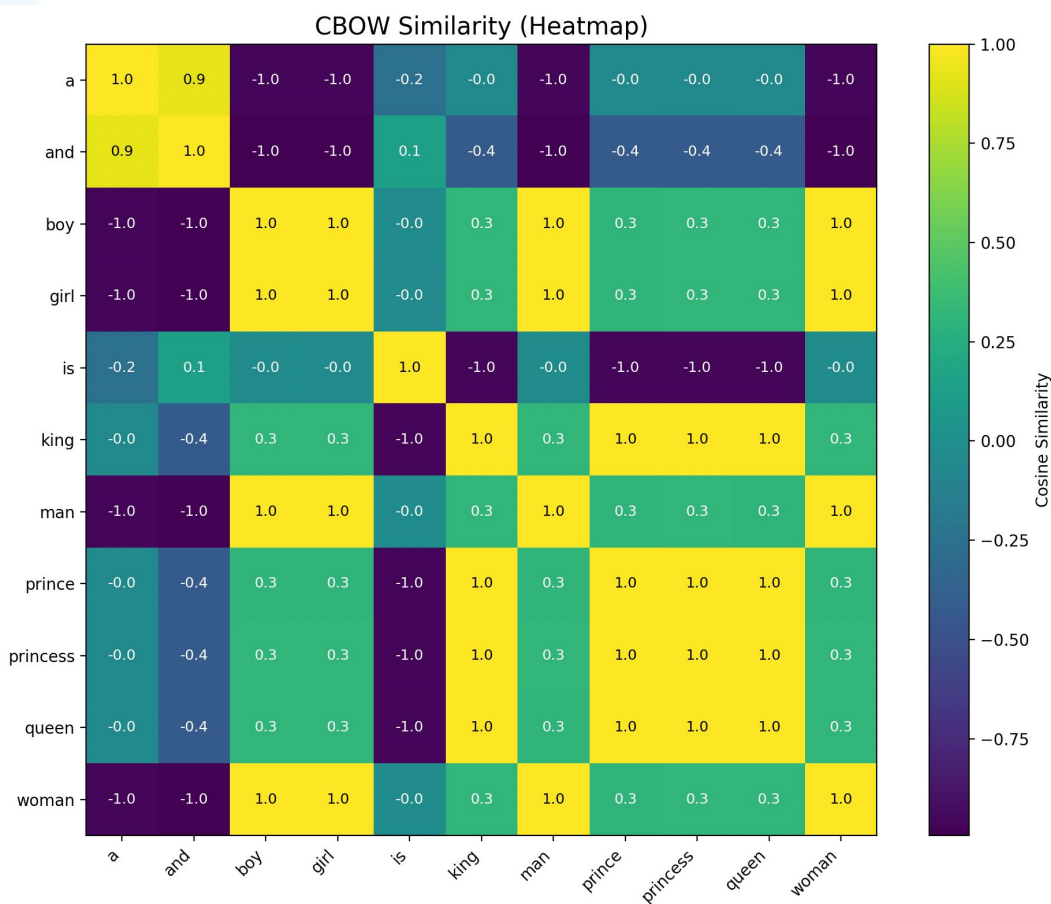
Visualization 1 - 2D Scatter Plot



3. CBOW

- Corresponding Implementation in Python:

Visualization 2 - Heatmap (Cosine Similarity)



Summary

4. Summary

Skip-gram Overview

- Predicts surrounding context words given a center word.
- Learns two matrices: input (center) embeddings and output (context) embeddings.
- Training pulls center vectors toward true context vectors and pushes them away from negatives.
- Strong for rare words but usually slower to train.

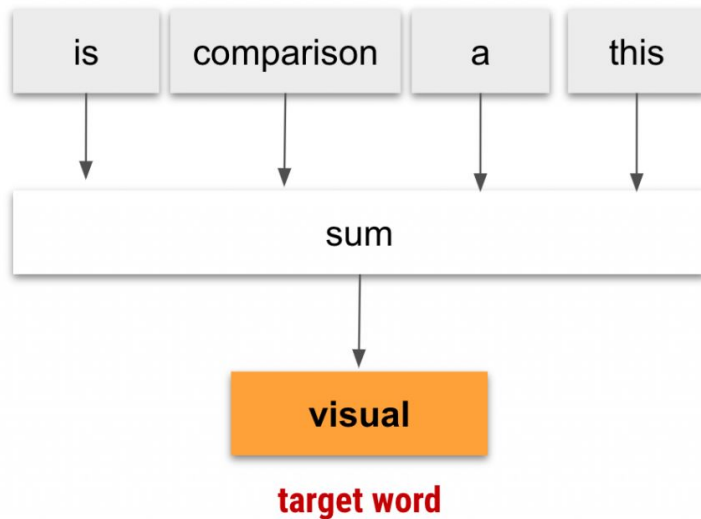
4. Summary

CBOW Overview

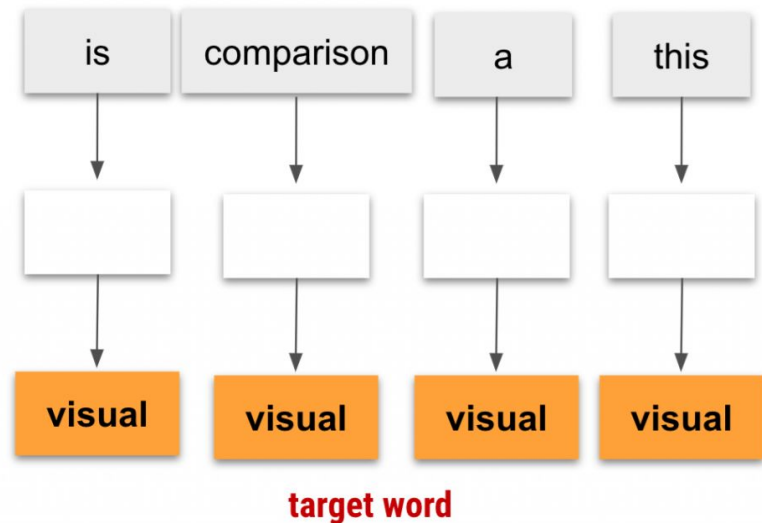
- Predicts the **target word from the average of context embeddings**.
- Trains faster and produces stable embeddings for **frequent words**.
- Less effective for rare words due to the averaging step, which smooths important distinctions.

4. Summary

CBOW



SkipGram



This is a visual comparison

4. Summary

Computation & Training Flow

- **Embedding lookup** (center/context \rightarrow vector).
- **Hidden projection** (CBOW: mean of context vectors; Skip-gram: center vector).
- **Dot-product logits** with output/context matrix.
- **Softmax \rightarrow class probabilities** (approximated by negative sampling in practice).
- **Negative log-likelihood(NLL) (cross-entropy) loss.**
- **SGD (or Adam) updates** to input/output embedding matrices.

4. Summary

Model Comparison

Aspect	CBOW	Skip-gram
Prediction	Target \leftarrow Context	Context \leftarrow Target
Speed	Faster	Slower
Best For	Frequent words	Rare words
Window	Small	Large
Data Requirement	Lower	Higher
Typical Use Cases	Classification; Sentiment analysis	Similarity; NER; Translation

4. Summary

Takeaways & Limitations

Takeaways

- Embeddings capture semantic structure in dense vector space
- Useful across domains: text, products, users, events
- Enable similarity search, analogy, clustering, and recommendation

Limitations

- Require sufficient data for stable vectors
- Static: cannot model polysemy
- Weak in morphology without subword modeling
- No contextual understanding (unlike **BERT**)

4. Summary

Improvements & Recommendations

For Practical Systems

- Increase embedding dimensionality
- Use Negative Sampling or Hierarchical Softmax
- Train on larger corpora

For Better Alternatives Representations

- using fastText for subword/morphology
- using BERT/ELMo for contextual meaning
- Combine embeddings with collaborative filtering for recommendations
- Evaluate using Precision@k, Recall@k, MAP

Thank you for listening

Q&A