📘 Title: CBOW (Continuous Bag of Words) - Full Walkthrough with Manual Math Explanation

---

🧠 Objective:

To manually demonstrate how CBOW (Continuous Bag of Words) learns word embeddings using simple sentence, step-by-step math, and logic suitable for Humanities and Social Sciences (HSS) students.

✍️ Example Sentence (Tiny Corpus):

"the cat sat on the mat"

Vocabulary: [the, cat, sat, on, mat, dog]

Index assignment:

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

We use:

- Embedding size = **2**
- Context window size = **1**

🎯 Task:

Given context words cat and on, predict the center word: sat

✅ Step 1: One-Hot Encode the Context Words

One-hot vector for "cat" (index 1): [0, 1, 0, 0, 0, 0] One-hot vector for "on" (index 3): [0, 0, 0, 1, 0, 0]

Add them:

[0, 1, 0, 0, 0, 0] + [0, 0, 0, 1, 0, 0] = [0, 1, 0, 1, 0, 0]

Average them:

[0, 1, 0, 1, 0, 0] / 2 = [0, 0.5, 0, 0.5, 0, 0]

This is our input vector.

✅ Step 2: Input Embedding Matrix W (6x2)

Each row corresponds to a word vector of size 2. These numbers are **randomly initialized** when training starts. They do not represent meaning yet. The training process will adjust them to become meaningful based on context.

Word	dim1	dim2
the	0.1	0.3
cat	0.2	0.4
sat	0.0	0.5
on	0.6	0.1
mat	0.3	0.7
dog	0.2	0.2

We retrieve vectors for cat and on:

- cat = [0.2, 0.4]
- on = [0.6, 0.1]

Average them:

([0.2 + 0.6] / 2, [0.4 + 0.1] / 2) = [0.4, 0.25] ← hidden layer output

---

✅ Step 3: Output Weight Matrix W' (2x6)

Transpose: now columns represent words

	the	cat	sat	on	mat	dog
dim1	0.3	0.2	0.4	0.6	0.1	0.7
dim2	0.5	0.3	0.2	0.4	0.6	0.2

Multiply [0.4, 0.25] with each column (dot product):

Word	Calculation	Output
the	0.4·(0.3) + 0.25·(0.5) = 0.12 + 0.125	0.245
cat	0.4·(0.2) + 0.25·(0.3) = 0.08 + 0.075	0.155
sat	0.4·(0.4) + 0.25·(0.2) = 0.16 + 0.05	0.21
on	0.4·(0.6) + 0.25·(0.4) = 0.24 + 0.1	0.34
mat	0.4·(0.1) + 0.25·(0.6) = 0.04 + 0.15	0.19
dog	0.4·(0.7) + 0.25·(0.2) = 0.28 + 0.05	0.33

✅ Step 4: Apply Softmax

Exponentiate all outputs (using e^x):

Word	e^score
the	e^0.245 ≈ 1.277
cat	e^0.155 ≈ 1.167
sat	e^0.210 ≈ 1.233
on	e^0.340 ≈ 1.405
mat	e^0.190 ≈ 1.209
dog	e^0.330 ≈ 1.391

Sum = 1.277 + 1.167 + 1.233 + 1.405 + 1.209 + 1.391 ≈ **7.682**

Now divide each by the total (normalize):

Word	Softmax Probability
the	1.277 / 7.682 ≈ 0.166
cat	1.167 / 7.682 ≈ 0.152
sat	1.233 / 7.682 ≈ 0.160 ← correct word!
on	1.405 / 7.682 ≈ 0.183
mat	1.209 / 7.682 ≈ 0.157
dog	1.391 / 7.682 ≈ 0.181

✅ Step 5: Compute Loss

We use **Cross-Entropy Loss** to measure how far our prediction is from the correct answer.

We predicted probabilities using softmax. The probability of the correct word "sat" was **0.160**.

So the loss becomes:

Loss = -log(probability of correct word)

= -log(0.160) ≈ 1.83

📌 **How do we know this prediction is wrong?**

- The original (raw) score before softmax for "sat" was **0.21** (from dot product).
- Softmax turned this into a probability: **0.160**.
- Since 0.160 is not very high (ideal = close to 1), it means the model isn't confident.

✅ Step 6: Backpropagation (with Gradient Descent)

🔁 **Backpropagation** helps the model learn from mistakes:

1. **Error Calculation**: For each word:

    - error = predicted_probability - actual
    - Only "sat" is the correct word (actual = 1), others = 0.
2. **Gradient Calculation**:
    - Gradients show how much each weight contributed to the error.
    - Computed for both W and W' matrices.
3. **Weight Updates**:
    - Each weight is adjusted using:

new_weight = old_weight - learning_rate * gradient

- This helps push the score for the correct word up (and wrong ones down).

Over many training steps, these weights (word vectors) become better at capturing the meaning/context of each word.
