CBOW (Continuous Bag-Of-Words) — Complete Notes

Zero-to-hero explanation, intuition, gradients, numeric examples, and practical tips

Purpose and scope

This document collects a full, self-contained explanation of the CBOW model: definitions, intuition, forward and backward computations, negative sampling, a worked numeric example, pseudocode, and practical hyperparameter advice. It also explicitly answers the common question: why are there two parameter matrices (V and U) and which one is used where.

1 Notation (keep this at top of your page)

- m (or V) = vocabulary size (number of unique words).
- d = embedding dimension (e.g. 50, 100, 300).
- C = number of context words (context window size). In notes above sometimes called m; here we use C to avoid confusion.
- K = number of negative samples (if using negative sampling).
- η = learning rate.
- $V \in \mathbb{R}^{m \times d}$ = input / embedding matrix. Row v_w is the embedding used when word w appears in the context / as input.
- $U \in \mathbb{R}^{m \times d}$ = output / classifier matrix. Row u_w is the vector used to score/predict word w as the center (the candidate output).
- One-hot vector for word i is e_i (length m with a 1 at position i).
- $\sigma(x) = 1/(1 + e^{-x})$ denotes the sigmoid function.
- For brevity, for a context summary h we write the score for word w as $z_w = u_w^\top h$.

2 Big-picture intuition

- CBOW predicts the center word from the surrounding context words. The intuition: "a word is known by the company it keeps" — so the model uses the neighbors as evidence for the missing middle word.
- The model maintains two matrices: V (how words look as inputs / context signals) and U (how words look when they're the thing to be predicted). They are two roles for the same word.
- Training moves vectors so that context summaries align with correct output vectors and misalign with incorrect ones.

3 CBOW model — forward pass (step-by-step)

Given one training example: context indices $c_1, \ldots, c_C$ (these are the words that surround a center position) and true center index o.

1. Lookup: for each context index c_i take row v_{c_i} from matrix V.
2. Pool: compute averaged context vector

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

   Averaging keeps the scale stable and makes the model invariant to the order of context words (bag-of-words).

3. Score: compute for each vocabulary word w the unnormalized score (logit)

$$z_w = u_w^\top h.$$

4. Softmax: produce probabilities

$$p(w \mid \text{context}) = \frac{\exp(z_w)}{\sum_{w' \in V} \exp(z_{w'})}.$$

5. Loss: negative log-likelihood (cross-entropy) for true center o:

$$L = -\log p(o \mid \text{context}).$$

4 Why two matrices V and U? (plain words)

- V stores how words behave as *inputs / context signals*. When a word appears in the left/right neighbors we fetch its row from V to form the summary h.
- U stores how words behave as *outputs / classifiers*. Each row of U is a linear classifier that asks: "does this context summary h match word w?" The dot product $u_w^\top h$ is that answer.
- Having both allows asymmetric modeling: how a word acts as input may differ from how it should be recognized as output. Training updates both sets of parameters.
- Common practice: after training use V rows as final embeddings (or average V and U).

5 Gradients — full softmax (exact formulas)

Define a probability vector p with components $p_w = p(w \mid \text{context})$. Let t be the one-hot vector for the true center o (i.e., $t_w = 1$ iff $w = o$). Then:

$$\frac{\partial L}{\partial u_w} = (p_w - t_w)\, h,$$
$$\frac{\partial L}{\partial h} = \sum_{w \in V} (p_w - t_w)\, u_w \;=\; U^\top (p - t).$$

Because $h = \frac{1}{C}\sum_i v_{c_i}$ we have for each context embedding

$$\frac{\partial L}{\partial v_{c_i}} = \frac{1}{C}\frac{\partial L}{\partial h}.$$

Intuition: the gradient on h distributes equally to the context word vectors.

6 Negative sampling objective (practical)

Full softmax requires summing over all m vocabulary words in the denominator. For large vocabularies this costs $O(m)$ per example. Negative sampling reduces that cost to $O(K)$ where K is small (5–20 typical).

Objective (for one training example): choose K negative samples $w_1, \ldots, w_K$ from noise distribution P_n (typically unigram$^{3/4}$). Then maximize (equivalently minimize negative):

$$L_{NS} = -\log \sigma(u_o^\top h) - \sum_{k=1}^{K} \log \sigma(-u_{w_k}^\top h).$$

Gradients (useful to implement):

$$\begin{aligned}
\frac{\partial L}{\partial u_o} &= (\sigma(u_o^\top h) - 1)\, h, \\
\frac{\partial L}{\partial u_{w_k}} &= \sigma(u_{w_k}^\top h)\, h \quad \text{(for each negative } w_k\text{)}, \\
\frac{\partial L}{\partial v_{c_i}} &= \frac{1}{C}\Big[(\sigma(u_o^\top h) - 1)\, u_o + \sum_{k=1}^{K} \sigma(u_{w_k}^\top h)\, u_{w_k}\Big].
\end{aligned}$$

Intuition: pull u_o toward h (positive), push negatives away; update context vectors so their average better supports u_o.

7 Numeric worked example (full softmax) — copy to paper

Vocabulary size $m = 5$, embedding dim $d = 3$. Context indices: 1 and 2 ($C = 2$).

```
Input embeddings V:
v1 = [0.2, 0.1, 0.0]
v2 = [0.1, 0.0, 0.2]

h = (v1 + v2)/2 = [0.15, 0.05, 0.10]

Output rows U:
u0 = [0.1, 0.0, 0.0]  -> score = 0.015
u1 = [0.0, 0.2, 0.1]  -> score = 0.020
u2 = [0.0, 0.1, 0.0]  -> score = 0.005
u3 = [0.3, 0.1, 0.2]  -> score = 0.070  (true center)
u4 = [0.2, 0.0, 0.1]  -> score = 0.040
```

```
softmax(exp(scores)) approx: [1.01511,1.02020,1.00501,1.07251,1.04081]
sum approx 5.15365 -> p(true center u3) ~= 1.07251/5.15365 ~= 0.2081
loss = -log(0.2081) ~ 1.57

gradients will push u3,h to have larger dot product and push others down

done.
```

8 Numeric worked example (negative sampling) — small

Use small $d = 2$ example to see updates (this mirrors earlier Skip-gram example but with averaged context). Suppose:

```
v_a = [0.5, 0.0]  (h = v_a if single context)
u_pos = [0.6, 0.1]
neg u1 = [-0.2, 0.5]
K = 1, lr = 0.5

s_pos = 0.6*0.5 + 0.1*0 = 0.30 -> sigma ~ 0.5744
s_neg = -0.2*0.5 + 0.5*0 = -0.10 -> sigma ~ 0.4750

grad_h = (1 - sigma(s_pos))*u_pos - sigma(s_neg)*u_neg
and update accordingly...
```

9 Pseudocode (CBOW with negative sampling)

```
Initialize V (m x d) and U (m x d) small random
for epoch in 1..epochs:
  for each position t in corpus:
    context = words in window around t (C words)
    h = average(V[context])          # shape (d,)
    pos = center index
    negatives = sample_noise(K)      # unigram^(3/4)
    # update outputs
    s_pos = dot(U[pos], h); sigma_pos = sigmoid(s_pos)
    U[pos] -= lr * (sigma_pos - 1) * h
    for w in negatives:
      s_w = dot(U[w], h); sigma_w = sigmoid(s_w)
      U[w] -= lr * (sigma_w) * h
    # update inputs (context embeddings)
    grad_h = (sigma_pos - 1) * U[pos] + sum( sigma_w * U[w] for w in negatives)
    for c in context:
      V[c] -= lr * (grad_h / len(context))
# After training use V rows (or (V+U)/2) as word embeddings
```

10 Hyperparameters and practical tips

- Window size: 2–5 typical for syntactic/substitutional similarity. Larger windows capture topic.
- Embedding dimension d: 50–300 common.
- Negative samples K: 5–20 typical.
- Negative sampling distribution: unigram$^{3/4}$ recommended.
- Subsampling frequent words: randomly drop high-frequency words to speed up and improve rare-word vectors.
- Learning schedule: start with modest lr (e.g. 0.05) and decay; original implementations used simple SGD with decay.
- Final embedding: use V rows, or averaged (V+U)/2.

11 Short FAQ (copyable answers)

Q: Why two matrices? A: V = input/context encodings; U = output/classifier vectors. They give the model flexibility and efficient updates.

Q: Does U change per forward pass? A: No, U is fixed during a single forward computation; it only changes when gradients are applied (during training). Different inputs produce different scores because h changes.

Q: What is being predicted in CBOW? A: The center word (the target) given the context words as input.

Q: What happens to each V row during update? A: A context V row receives a small gradient that moves it so that its contribution to the average h makes the true center score higher in future.

Closing

This file collects CBOW intuition, math, code, and examples. If you want extra numeric backward arithmetic (component-wise numbers for a single negative-sampling update) I can add an additional worked block.