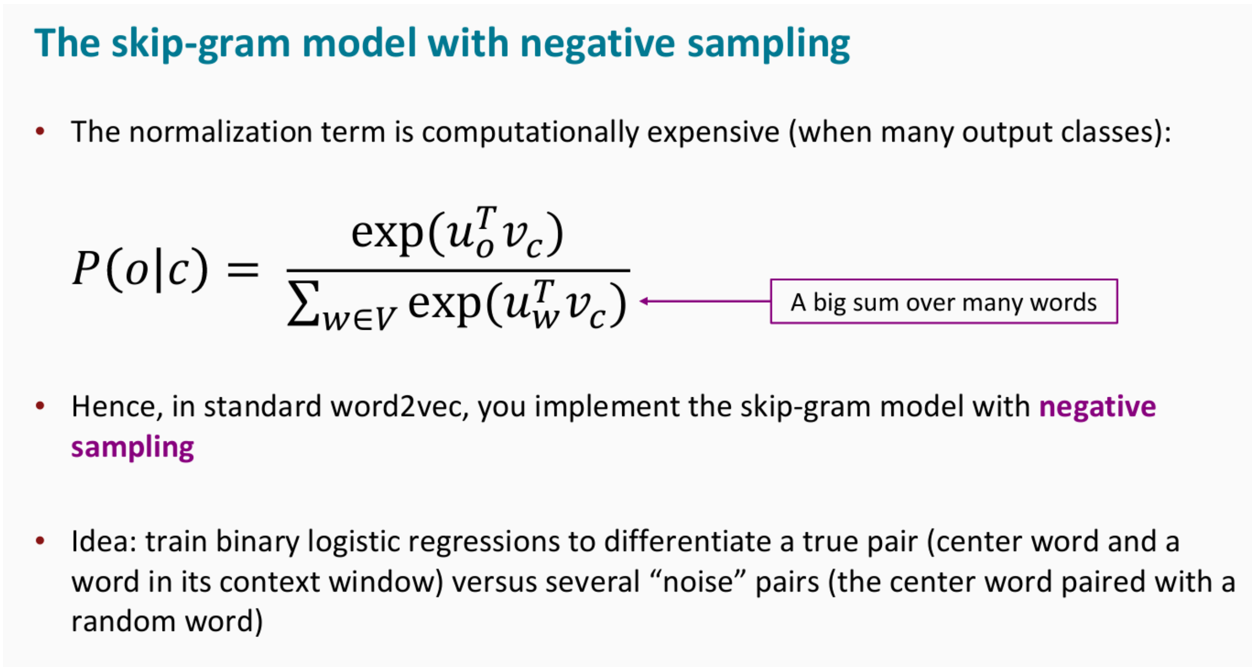


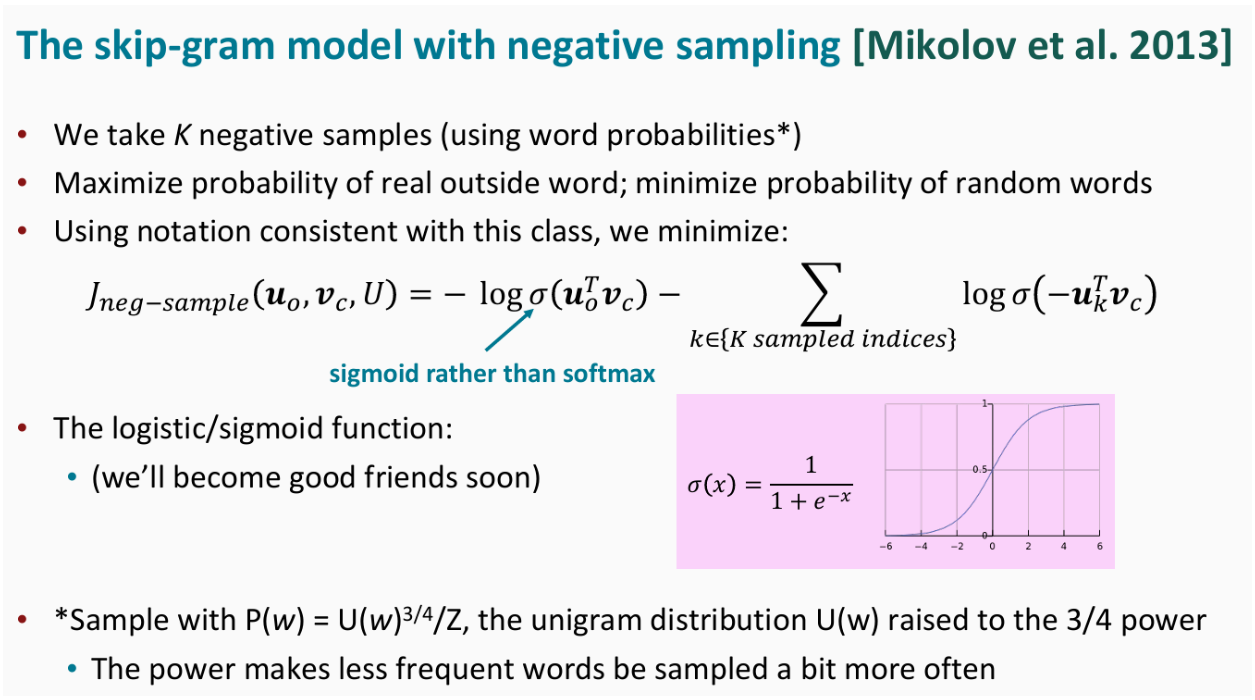
The skip-gram model with negative sampling

- The normalization term is computationally expensive (when many output classes):

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \longleftarrow \boxed{\text{A big sum over many words}}$$

- Hence, in standard word2vec, you implement the skip-gram model with **negative sampling**

- Idea: train binary logistic regressions to differentiate a true pair (center word and a word in its context window) versus several “noise” pairs (the center word paired with a random word)



The skip-gram model with negative sampling [Mikolov et al. 2013]

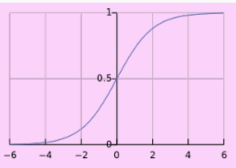
- We take *K* negative samples (using word probabilities*)
- Maximize probability of real outside word; minimize probability of random words
- Using notation consistent with this class, we minimize:

$$J_{neg-sample}(\boldsymbol{u}_o, \boldsymbol{v}_c, U) = -\log \sigma(\boldsymbol{u}_o^T \boldsymbol{v}_c) - \sum_{k \in \{K \text{ sampled indices}\}} \log \sigma(-\boldsymbol{u}_k^T \boldsymbol{v}_c)$$

sigmoid rather than softmax

- The logistic/sigmoid function:
  - (we’ll become good friends soon)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



- *Sample with P(*w*) = U(*w*)$^{3/4}$/Z, the unigram distribution U(w) raised to the 3/4 power
  - The power makes less frequent words be sampled a bit more often

In the standard softmax setup, the denominator requires summing over **every word in the vocabulary**—which could mean calculating dot products and exponentials for **hundreds of thousands of words**. That’s computationally expensive.

Negative sampling offers a shortcut: instead of computing the full softmax, we train a set of **small logistic regression tasks**. For each real (center, context) word pair, the model is trained to score it highly, while **randomly sampled “negative” words** are scored low. This is the core idea behind

Skip-gram with Negative Sampling (SGNS)—efficiently teaching the model which words *should* and *shouldn't* appear together.

✅ Skip-Gram with Negative Sampling (SGNS) – Explained Simply

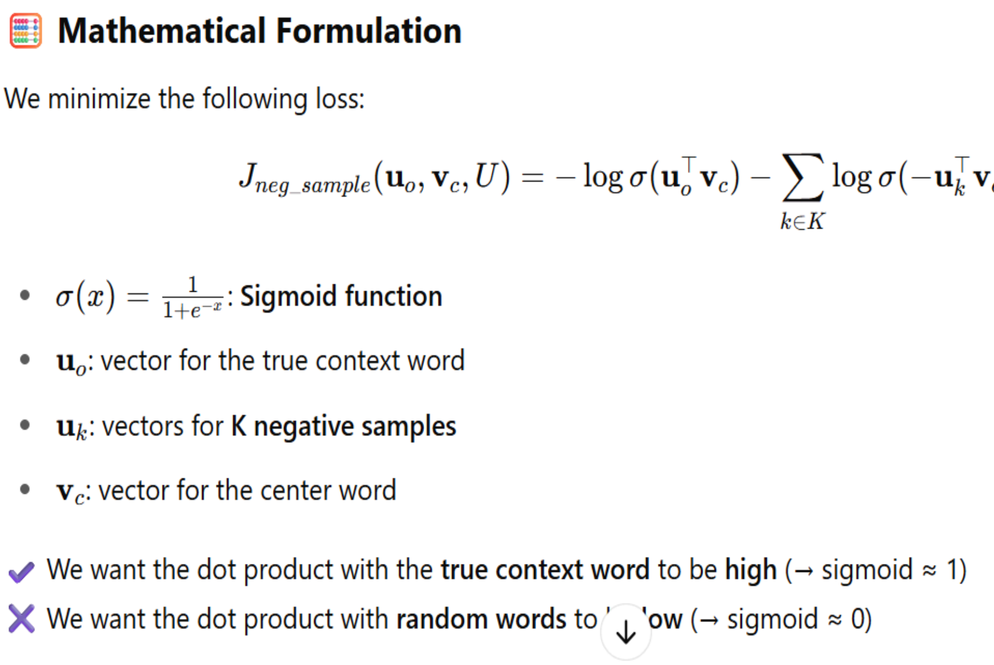
In the **standard softmax**, we had to compute probabilities over all words in the vocabulary, which is very expensive (e.g., 400,000 words).
To make this faster, **Negative Sampling** simplifies the task.

🧠 What's the Idea?

- For each **real word pair** (center word and actual context word), we:
  - **maximize** the probability that they appear together.
- For a few **randomly chosen "negative" words**, we:
  - **minimize** the probability that they appear with the center word.

This creates a **binary classification problem**:

- Is this word a true context word (positive sample)?
- Or a random one (negative sample)?



🧮 Mathematical Formulation

We minimize the following loss:

$$J_{neg\_sample}(\mathbf{u}_o, \mathbf{v}_c, U) = -\log \sigma(\mathbf{u}_o^\top \mathbf{v}_c) - \sum_{k \in K} \log \sigma(-\mathbf{u}_k^\top \mathbf{v}_c)$$

- $\sigma(x) = \frac{1}{1+e^{-x}}$: Sigmoid function
- $\mathbf{u}_o$: vector for the true context word
- $\mathbf{u}_k$: vectors for K **negative samples**
- $\mathbf{v}_c$: vector for the center word

✔️ We want the dot product with the **true context word** to be **high** (→ sigmoid ≈ 1)
❌ We want the dot product with **random words** to **[illegible]ow** (→ sigmoid ≈ 0)

⚖️ Sampling Negative Words

- We don't choose negative words uniformly.
- We use the **unigram distribution** raised to the **3/4 power**:

$$P(w) = \frac{U(w)^{3/4}}{Z}$$

- This means **frequent words (like "the") are downsampled a bit**, and **less frequent words are sampled more often**.
- This creates a balance between completely uniform and frequency-based sampling.

🔍 Why the Sigmoid Instead of Softmax?

- **Softmax** needs computation over the whole vocabulary → expensive.
- **Sigmoid** is simpler: it gives a probability for **each pair independently**.

🗣️ **Analogy:**

Imagine training the model to recognize "real friends" (true word pairs) and "random strangers" (negative samples). You want it to confidently pick out friends from the crowd, without needing to scan every single stranger.

✅ Why Use the Unigram Distribution in Negative Sampling?

1. Reflects Word Frequency

- **The unigram distribution U(w)U(w)U(w) assigns a probability to each word based on how often it occurs in the corpus.**
- **This means common words like "the", "is", and "and" are sampled more often than rare words.**

2. Improves Learning Efficiency

- **Frequent words are more likely to appear as negatives, so the model learns to ignore them appropriately.**
- **Sampling negatives that never occur in real contexts (e.g., rare technical terms) doesn't help the model much.**

3. But Pure Frequency Isn't Optimal

- **Very frequent words dominate (like “the”), which can overwhelm training.**
- **So we adjust the unigram distribution to balance frequency and informativeness.**

🔧 **Adjustment: Raising to the 3/4 Power**

$$P(w) = \frac{U(w)^{3/4}}{Z}$$

- **This smooths the distribution:**
  - **Reduces dominance of very frequent words.**
  - **Boosts presence of less frequent but still useful words.**
- **It's a compromise between:**
  - **Uniform sampling (equal chance for all)**
  - **Raw frequency sampling (pure unigram)**

🧠 **In Summary:**

We use the unigram distribution (raised to the 3/4 power) to:

- **Ensure negative samples are realistic and informative,**
- **Avoid over-sampling very common words,**
- **And improve the quality of learned word vectors.**

❓Do we check all real (center, context) pairs over the entire vocabulary?

No. We do not evaluate every possible (center, context) word pair in the vocabulary.

✅ Here’s What Actually Happens:

1. Real Word Pairs (Positive Samples)

- For each position in the corpus, we look at a center word and its surrounding context words within a fixed window (e.g., ±2).
- These are the real (center, context) pairs.
- We do use all of these real pairs during training — they're part of the dataset.
  → But only a few pairs per training step, based on the sliding window.

🧠 Think: One training step = one center word and its real context words.

---

2. Negative Word Pairs (Negative Samples)

- For each real (center, context) pair, we randomly sample kkk words from the vocabulary (using the adjusted unigram distribution raised to the 3/4 power).
- These are fake pairs: (center, negative word), which are not real context words.
- The model learns to assign low probability to these.

✅ So instead of computing over all 40 million words, we only check a small number of negative samples, like k=5k = 5k=5 to 202020.

💡 Why Is This Efficient?

- Sigmoid function works on individual word pairs.
- You only compute:
  - One positive pair: $\boldsymbol{\sigma}(u^T_{context}\, v_{center})$
  - k negative pairs:$\boldsymbol{\sigma}(-u^T_{neg}\, v_{center})$
- So each training step involves 1 + k dot products and sigmoid operations—not millions.

🔁 Summary of Training Step in SGNS:

For each center word:

- Use real context words (within window) → positive samples
- Sample k fake words from unigram$^{3/4}$ → negative samples
- Apply sigmoid to each pair → compute loss → update embeddings via gradient descent

---

🔍 Your Key Insight:

> “As sigmoid gives a probability for each pair independently, it is less expensive”

✅ Correct. That’s exactly why negative sampling + sigmoid is used instead of softmax.