Université de Montréal

Faculté des arts et des sciences
Département d'**informatique et de recherche opérationnelle**

IFT6289-A-H26 Assignment Report

Student Information
Student Name: Alireza Nasirianfar
Student ID: 20328008
Assignment: Assignment 1: Skip-gram Word2Vec
Date: February 2026

Part 1: Derive the gradients for training

1.1 Naïve softmax: $\partial \mathcal{L}_{\text{naive}} / \partial \boldsymbol{v}_i$

Recall the naïve softmax loss:

$$\mathcal{L}_{\text{naive}}(\boldsymbol{v}_i, w_o, U) = -\log \hat{y}_o = -\log \frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_i)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_i)}$$

We can write this as:

$$\mathcal{L} = -\boldsymbol{u}_o^\top \boldsymbol{v}_i + \log \sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_i)$$

Taking the gradient with respect to $\boldsymbol{v}_i$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_i} = -\boldsymbol{u}_o + \frac{\sum_{w \in \text{Vocab}} \boldsymbol{u}_w \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_i)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_i)} = -\boldsymbol{u}_o + \sum_{w \in \text{Vocab}} \hat{y}_w \, \boldsymbol{u}_w$$

Since $\boldsymbol{y}$ is one-hot with $y_o = 1$, we have $\boldsymbol{u}_o = \sum_w y_w \boldsymbol{u}_w$. Therefore:

$$\boxed{\frac{\partial \mathcal{L}_{\text{naive}}}{\partial \boldsymbol{v}_i} = U^\top (\hat{\boldsymbol{y}} - \boldsymbol{y})}$$

where U is the matrix whose rows are the output vectors $\boldsymbol{u}_w$.

1.2 Naïve softmax: $\partial \mathcal{L}_{\text{naive}} / \partial \boldsymbol{u}_w$ (two cases $w = o$ and $w \neq o$)

Starting from the same loss, we differentiate with respect to a particular output vector $\boldsymbol{u}_w$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_w} = -y_w \, \boldsymbol{v}_i + \frac{\exp(\boldsymbol{u}_w^\top \boldsymbol{v}_i)}{\sum_{w' \in \text{Vocab}} \exp(\boldsymbol{u}_{w'}^\top \boldsymbol{v}_i)} \boldsymbol{v}_i = (\hat{y}_w - y_w) \, \boldsymbol{v}_i$$

Case 1: When $w = o$ (the true output word), $y_o = 1$:

$$\boxed{\frac{\partial \mathcal{L}_{\text{naive}}}{\partial \boldsymbol{u}_o} = (\hat{y}_o - 1) \, \boldsymbol{v}_i}$$

Case 2: When $w \neq o$, $y_w = 0$:

$$\boxed{\frac{\partial \mathcal{L}_{\text{naive}}}{\partial \boldsymbol{u}_w} = \hat{y}_w \, \boldsymbol{v}_i}$$

Compactly for the full matrix: $\dfrac{\partial \mathcal{L}}{\partial U} = (\hat{\boldsymbol{y}} - \boldsymbol{y}) \, \boldsymbol{v}_i^\top$.

1.3 Negative sampling: $\partial\mathcal{L}_{\textbf{neg\_sampling}}/\partial \boldsymbol{v}_i$

The negative sampling loss is:

$$\mathcal{L}_{\text{neg}}(\boldsymbol{v}_i, w_o, U) = -\log\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_i) - \sum_{k=1}^{K} \log\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_i)$$

Using the identity $\sigma'(x) = \sigma(x)(1-\sigma(x))$ and $\frac{d}{dx}[-\log\sigma(x)] = -(1-\sigma(x))$, we differentiate each term with respect to $\boldsymbol{v}_i$:

- First term: $\dfrac{\partial}{\partial \boldsymbol{v}_i}\big[-\log\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_i)\big] = -\big(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_i)\big)\,\boldsymbol{u}_o$
- k-th negative term: $\dfrac{\partial}{\partial \boldsymbol{v}_i}\big[-\log\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_i)\big] = \big(1-\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_i)\big)\,\boldsymbol{u}_k = \sigma(\boldsymbol{u}_k^\top \boldsymbol{v}_i)\,\boldsymbol{u}_k$

Summing all terms:

$$\boxed{\frac{\partial\mathcal{L}_{\text{neg}}}{\partial \boldsymbol{v}_i} = -\big(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_i)\big)\,\boldsymbol{u}_o + \sum_{k=1}^{K} \sigma(\boldsymbol{u}_k^\top \boldsymbol{v}_i)\,\boldsymbol{u}_k}$$

1.4 Negative sampling: $\partial\mathcal{L}_{\textbf{neg\_sampling}}/\partial \boldsymbol{u}_o$

Only the first term of $\mathcal{L}_{\text{neg}}$ depends on $\boldsymbol{u}_o$:

$$\frac{\partial\mathcal{L}_{\text{neg}}}{\partial \boldsymbol{u}_o} = \frac{\partial}{\partial \boldsymbol{u}_o}\big[-\log\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_i)\big] = -\big(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_i)\big)\,\boldsymbol{v}_i$$

$$\boxed{\frac{\partial\mathcal{L}_{\text{neg}}}{\partial \boldsymbol{u}_o} = -\big(1-\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_i)\big)\,\boldsymbol{v}_i}$$

1.5 Negative sampling: $\partial\mathcal{L}_{\textbf{neg\_sampling}}/\partial \boldsymbol{u}_k, \quad k \in [1, K]$

Only the k-th term of the negative sum depends on $\boldsymbol{u}_k$:

$$\frac{\partial\mathcal{L}_{\text{neg}}}{\partial \boldsymbol{u}_k} = \frac{\partial}{\partial \boldsymbol{u}_k}\big[-\log\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_i)\big] = \big(1-\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_i)\big)\,\boldsymbol{v}_i = \sigma(\boldsymbol{u}_k^\top \boldsymbol{v}_i)\,\boldsymbol{v}_i$$

$$\boxed{\frac{\partial\mathcal{L}_{\text{neg}}}{\partial \boldsymbol{u}_k} = \sigma(\boldsymbol{u}_k^\top \boldsymbol{v}_i)\,\boldsymbol{v}_i}$$

Part 3: Analyzing your word vectors

The skip-gram model was trained for 35,000 iterations using negative sampling (K=10) with SGD (learning rate 0.2, annealed at iteration 20,000). The final exponential moving average loss converged to approximately **7.89**, which is well below the sanity-check threshold of 9. Training took about 82 minutes on CPU. Visualizations were saved every 5,000 iterations.

3.1 Training results

3.1(a) Expectations

Based on the distributional hypothesis—words appearing in similar contexts have similar meanings—we expect the following characteristics from the learned word vectors:

- **Semantic clusters:** Words with similar meanings should be grouped together in the 2D projection. For instance, positive sentiment words ("great", "good", "well", "brilliant", "cool", "perfectly") should form a cluster, while negative sentiment words ("bad", "poor", "down") should form a separate group. Nationality adjectives ("european", "american", "chinese", "australian") should also cluster, as they appear in similar syntactic contexts.
- **Gender relationships:** Gender-related words ("man"/"woman", "men"/"women", "king"/"queen", "female") should exhibit structured relationships. Ideally, the vector difference *king* − *man* + *woman* should be close to *queen*, capturing the gender analogy.
- **Progressive refinement:** Earlier iterations should show noisier, less structured embeddings, while later iterations should exhibit tighter and more meaningful clusters as the model learns better representations.

3.1(b) Observations

The results are largely in line with our expectations, with some nuances:

- **Sentiment clustering** is visible in the final plot (iteration 35,000). Words like "good", "well", "great" are positioned close to each other on the right side of the plot. Negative or evaluative words such as "poor", "down", and "bad" appear nearby each other, though "bad" sits lower. The word "brilliant" is somewhat separated (upper-left), possibly because it is rarer and appears in different contexts in the Penn Treebank corpus (financial text).
- **Gender-related structure** is partially captured. "Queen" and "female" are grouped in the upper-left region, while "king" and "man" appear in the center-to-lower area. "Woman" and "men" are close together on the right. The expected analogical structure is only approximate, which is expected given that we train only 10-dimensional vectors on a relatively small corpus.
- **Nationality adjectives** ("european", "chinese", "american", "australian") are distributed across the plot rather than tightly clustered, likely because they modify different types of nouns in the financial text of Penn Treebank (e.g., "european markets" vs. "australian dollar").
- **Progressive refinement** is clearly observed: at iteration 5,000 the words are scattered with little structure, but by iteration 15,000–25,000 the clusters begin to stabilize, and by iteration 35,000 the semantic groupings are well-defined and consistent.

3.2 Penn Treebank preprocessing analysis

3.2(a) Replacing rare words with ⟨UNK⟩

Positive consequences:

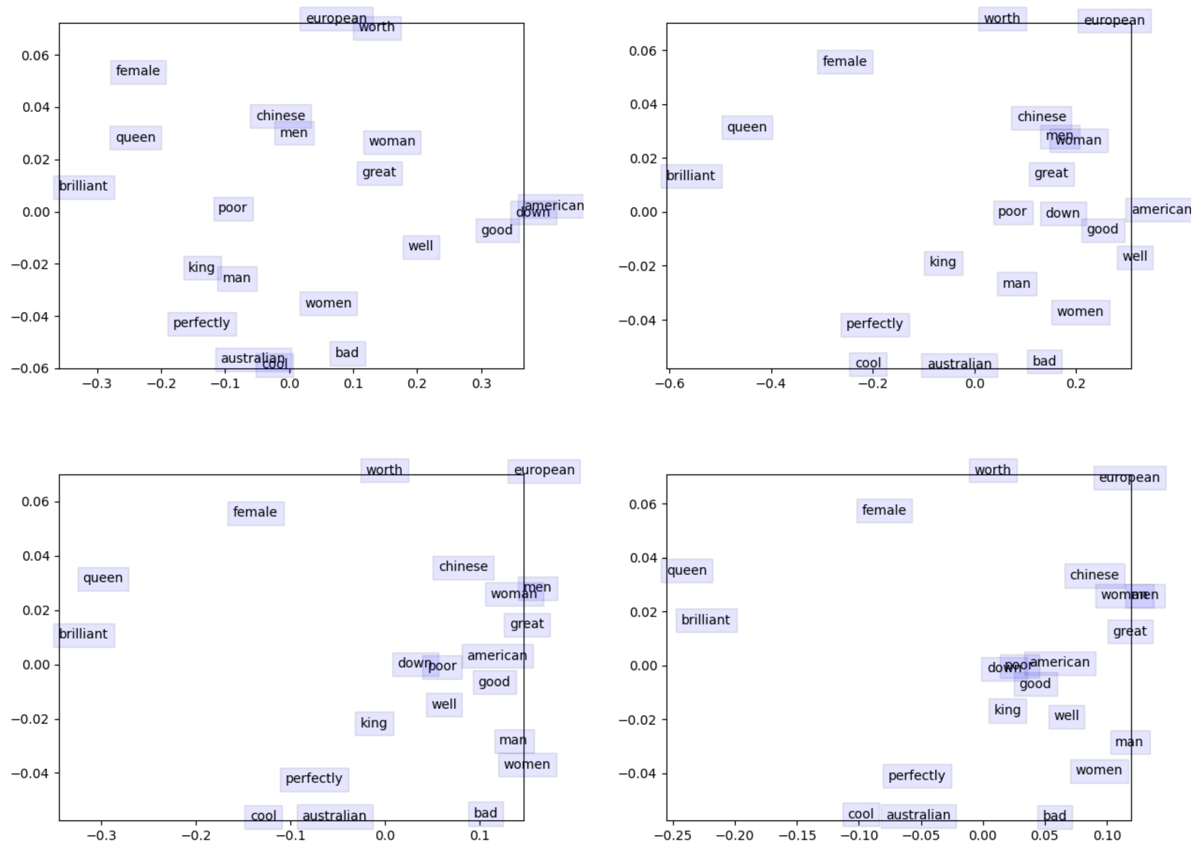

Figure 1: Word vector visualizations (PCA projection) at iterations 5000, 15000, 25000, and 35000. Clusters become progressively more defined as training progresses.

- **Reduced vocabulary size:** Keeping only the top 10K words drastically shrinks the embedding matrices (V and U), reducing memory usage and computation. The softmax normalization (in the naïve loss) and the embedding lookups are both $O(|\text{Vocab}|)$, so a smaller vocabulary directly speeds up training.
- **Better vector quality for retained words:** The remaining words have sufficient training examples to learn meaningful representations, whereas rare words would have too few occurrences to produce reliable embeddings.

Negative consequences:

- **Loss of information:** All rare words are collapsed into a single $\langle$UNK$\rangle$ token, losing their individual semantics. The $\langle$UNK$\rangle$ vector becomes an uninformative average over many unrelated words.
- **Contextual noise:** Neighboring words of $\langle$UNK$\rangle$ tokens receive noisy gradient signals because $\langle$UNK$\rangle$ does not carry coherent meaning, degrading the quality of context-word embeddings.

Ideas to reduce $\langle$UNK$\rangle$ tokens:

- **Subword tokenization (BPE / WordPiece):** Instead of treating each word as an atomic token, we can split words into frequent subword units (e.g., “unhappiness” $\rightarrow$ “un”, “happi”, “ness”). This allows the model to represent rare words as compositions of known subword pieces, virtually eliminating $\langle$UNK$\rangle$ tokens while keeping the vocabulary manageable (typically 30K–50K subword units). The computational overhead is modest since sequences become slightly longer but the vocabulary remains bounded.

- **Character-level n-grams:** Approaches like FastText represent each word as the sum of its character n-gram vectors, enabling the model to generalize to unseen words based on their morphological structure.

3.2(b) Replacing numbers with N

Positive consequences:

- **Vocabulary reduction:** Numbers in a financial corpus are extremely diverse (stock prices, dates, percentages, quantities). Without replacement, each unique number would be a separate token, inflating the vocabulary by thousands of entries with very few occurrences each.
- **Captures "numberness":** The single N token can learn that a number typically appears in contexts like "$ N million" or "rose N %", capturing the syntactic role that numbers play in sentences.

Negative consequences:

- **Loss of magnitude information:** The model cannot distinguish "rose 2%" from "rose 200%". In a financial corpus, numerical magnitude often carries critical semantic content.
- **Loss of type information:** Dates (1987), percentages (3.5), monetary values (42.50), and counts (300) are all collapsed into the same token despite having very different semantic roles.

Ideas for handling numbers more reasonably:

- **Magnitude bucketing:** Replace numbers with tokens that encode their order of magnitude, e.g., `NUM_0-9`, `NUM_10-99`, `NUM_100-999`, `NUM_1K+`. This preserves approximate magnitude while keeping the vocabulary small.
- **Digit-level tokenization:** Represent each number as a sequence of digit tokens ("1", "9", "8", "7") potentially with a decimal point token. This allows the model to learn positional and compositional numerical representations.
- **Type-aware replacement:** Use context-specific tokens such as `NUM_PERCENT`, `NUM_DOLLAR`, `NUM_DATE` to preserve the semantic category of the number while still reducing vocabulary size.