Learning to Reason with Small Models - Report

Viktor Ćurčić

Abstract

This report presents an analysis of the Word2Vec model with negative sampling, developed without relying on deep learning frameworks, implemented from scratch. The implementation covers the complete training pipeline, including data preprocessing, forward computation, loss formulation, gradient derivation and parameter updates.

1 Task Definition

The goal of this task is to implement the core training loop of the *word2vec* model using only NumPy, without relying on any deep learning frameworks such as PyTorch or TensorFlow. The implementation must include the complete optimization procedure: forward pass, loss computation, gradient derivation and parameter updates.

The chosen variant is the skip-gram with negative sampling model. Given a sequence of tokens extracted from a text corpus, the model learns dense vector representations (embeddings) of words such that words occurring in similar contexts have similar vector representations. The training data consists of pairs (w_c, w_o), where w_c is a center word and w_o is a context word sampled from a window around the center word in the original text.

Formally, let V denote the vocabulary size and d the embedding dimension. The model maintains two embedding matrices $\mathbf{W}_{\text{in}}$ and $\mathbf{W}_{\text{out}}$ where each row of $\mathbf{W}_{\text{in}}$ represents the embedding of a center word and each row of $\mathbf{W}_{\text{out}}$ represents the embedding of a context word.

2 Core training loop

2.1 Model Formulation

The model first assigns each word a vector and checks how similar the vectors of two nearby words are. The compatibility score between the center and context word is computed as the dot product:

$$s = \mathbf{v}_c^\top \mathbf{v}_o.$$

Where $\mathbf{v}_c$ and $\mathbf{v}_o$ are learned numerical vectors representing each word in the vocabulary.
This score is converted into a probability using the sigmoid function:

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

The positive sample probability is therefore:

$$p_{\text{pos}} = \sigma(\mathbf{v}_c^\top \mathbf{v}_o).$$

This value can be interpreted as the model's confidence that the two words should appear close to each other in the text.

To avoid computing a full softmax over the entire vocabulary, which would be computationally demanding for large V, negative sampling is used. For every correct word pair, the model also picks a few random "wrong" words to act as negative examples. These negatives are chosen more often if they appear frequently in the text, which helps the model learn better word representations. The model then checks how similar each of these wrong words is to the center word and learns to give them low similarity scores. In the implementation, this distribution is proportional to the unigram frequencies raised to the power of 0.75, which empirically improves the quality of learned embeddings.

For each negative sample w_{n_k} with embedding $\mathbf{v}_{n_k}$, the model computes:

$$p_{\text{neg},k} = \sigma(\mathbf{v}_{n_k}^\top \mathbf{v}_c).$$

The loss for a single training pair is:

$$\mathcal{L} = -\log\left(\sigma(\mathbf{v}_c^\top \mathbf{v}_o)\right) - \sum_{k=1}^{K} \log\left(1 - \sigma(\mathbf{v}_{n_k}^\top \mathbf{v}_c)\right).$$

The first term encourages the model to assign high similarity to true center–context pairs, while the second term penalizes high similarity between the center word and randomly sampled noise words.

2.2 Gradient Derivation and Parameter Updates

After the model predicts how likely the true word pair is and how likely the negative (wrong) pairs are, it measures how wrong those predictions were. It then slightly adjusts the word vectors so that real word pairs become more similar and random word pairs become less similar. This adjustment is done step by step using gradient descent, with small updates to keep training stable.

Let:

$$p_{\text{pos}} = \sigma(\mathbf{v}_c^\top \mathbf{v}_o), \quad p_{\text{neg},k} = \sigma(\mathbf{v}_{n_k}^\top \mathbf{v}_c).$$

The gradient of the loss with respect to the positive score is:

$$\frac{\partial \mathcal{L}}{\partial(\mathbf{v}_c^\top \mathbf{v}_o)} = p_{\text{pos}} - 1.$$

Similarly, for each negative sample:

$$\frac{\partial \mathcal{L}}{\partial(\mathbf{v}_{n_k}^\top \mathbf{v}_c)} = p_{\text{neg},k}.$$

Using the chain rule, the gradient with respect to the center embedding is:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}_c} = (p_{\text{pos}} - 1)\mathbf{v}_o + \sum_{k=1}^{K} p_{\text{neg},k}\mathbf{v}_{n_k}.$$

The gradient with respect to the context embedding is:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}_o} = (p_{\text{pos}} - 1)\mathbf{v}_c.$$

For each negative embedding:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}_{n_k}} = p_{\text{neg},k}\mathbf{v}_c.$$

The parameters are updated using stochastic gradient descent:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}},$$

where η is the learning rate.

In the implementation, numerical stability is ensured by clipping the input to the sigmoid function to a fixed range. This prevents overflow of the exponential function during training.

2.3 Training Pipeline and Engineering Choices

The training corpus is first tokenized and filtered by removing common stop words. A vocabulary is constructed by discarding rare words below a minimum frequency threshold. To reduce the dominance of extremely frequent words, subsampling is applied: each word is discarded according to its relative frequency. This reduces the number of trivial training pairs and improves the quality of learned embeddings.

Context windows are sampled with a random radius, which introduces variability in the training signal and experimentally improves generalization. Negative samples are drawn from a precomputed cumulative distribution to allow efficient sampling.

More on the engineering choices will be discussed in section 6.

2.4 Evaluation

After training, the learned embeddings are normalized and evaluated using cosine similarity. For a given query word, the most similar words are retrieved by computing:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^\top \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}.$$

Qualitative inspection shows that semantically related words (e.g., animals, actions, descriptive adjectives) cluster together in the embedding space. This confirms that the implemented optimization procedure successfully captures distributional semantics from raw text using only NumPy-based operations.

3 Implementation Challenges

During the development of the Word2Vec training pipeline, several implementation issues were encountered. These issues were not related to syntax or basic programming errors, but to subtle mistakes in the mathematical formulation of the optimization procedure and to practical engineering constraints. Resolving these problems significantly improved both the stability of training and the semantic quality of the learned embeddings.

3.1 Gradient Sign Error in Negative Sampling

The most critical issue was an incorrect gradient formulation for the negative sampling objective. The skip-gram with negative sampling loss for a single training pair is defined as

$$\mathcal{L} = -\log \sigma(\mathbf{v}_c^\top \mathbf{v}_o) - \sum_{k=1}^{K} \log\left(1 - \sigma(\mathbf{v}_c^\top \mathbf{v}_{n_k})\right),$$

where $\mathbf{v}_c$ refers to the center word embedding, $\mathbf{v}_o$ the true context embedding and $\mathbf{v}_{n_k}$ the embeddings of negatively sampled noise words.

An initial implementation error applied the sigmoid function with the wrong sign in the backward pass for negative samples. As a result, the gradients encouraged both positive and negative word pairs to move in similar directions in the embedding space. This led to a collapse of the embeddings: most vectors had cosine similarities close to one for almost all word pairs. At the same time, the training loss increased instead of decreasing, indicating that the optimization dynamics were incorrect.

After correcting the gradient expressions so that negative samples contributed gradients proportional to $\sigma(\mathbf{v}_c^\top \mathbf{v}_{n_k})$, the embeddings recovered meaningful structure. The loss began to decrease monotonically across epochs and queries in the embedding space started to return semantically related words rather than arbitrary tokens. This debugging step showcased the sensitivity of representation learning to small sign errors in the objective function and reinforced the importance of explicitly deriving and validating gradients.

3.2 Numerical Stability

A second issue concerned numerical stability in the sigmoid function. Large dot products between embeddings can cause overflow in the exponential function, which in turn leads to invalid values during training. To address this, the input to the sigmoid function was clipped to a fixed range. This modification stabilized training and prevented exploding or vanishing gradients in later epochs.

In addition, the learning rate was decayed across epochs. With a fixed learning rate, early training progress was fast but later updates became noisy and unstable. A simple exponential decay schedule produced smoother convergence in the loss curve.

3.3 Subsampling

Subsampling of frequent words was implemented to reduce the dominance of extremely common tokens. Without subsampling, a large fraction of training pairs consisted of trivial high-frequency words, which slowed down convergence and reduced the diversity of training signals. After introducing subsampling, the effective training set became more informative and the learned embeddings exhibited clearer semantic

clusters. For the words such as ”happy”, most often the cosine similarity returned the words ”as” and ”is” as the nearest neighbors, since they appeared together as word-pairs. This changed prompted the model to look for more appropriate pairs.

During early experiments, overly aggressive subsampling and vocabulary filtering resulted in very few valid training pairs being generated. This led to near-zero parameter updates - vanishing gradients. Adjusting the minimum frequency threshold and subsampling parameters restored a sufficient number of training examples and significantly improved training dynamics.

3.4 Use of a Reduced Dataset

The TinyStories corpus was used as the training dataset. However, training on the full dataset would have been prohibitively time-consuming in a pure NumPy implementation, especially given the nested loops in the skip-gram training procedure. To remain within reasonable computational limits, only a subset of the dataset was used.

This choice reflects a deliberate trade-off between dataset size and computational cost. The primary objective of the task was to correctly implement and analyze the optimization procedure of Word2Vec rather than to achieve complete embedding quality. Training on a reduced subset allowed rapid iteration, debugging of gradient computations and inspection of convergence behavior, while still producing meaningful embeddings.

Because of the reduced set size, not all meaningful semantic similarities were captured. For the words such as ”run”, the model found jump, fetch, chase, roll, basketball as the most similar, showing its ability do discern actions and verbs. As for the word ”big”, breaths, mound, aboard, raking, smile were found, which shows the ability of the model to connect the phrases (”big smile”), but not other semantic similarities. With more data, this could be improved.

3.5 Loss Curves

The training process was monitored using the average loss per epoch, which was plotted over time. Before correcting the gradient formulation, the loss increased and exhibited unstable behavior, indicating divergence. After fixing the negative sampling gradients and stabilizing the sigmoid computation, the loss curve showed a consistent downward trend across epochs. This provided empirical confirmation that the optimization procedure was functioning as intended.

The loss curve also served as a diagnostic tool during development. Changes to subsampling rates, learning rate schedules and dataset size were reflected in the shape of the curve, allowing identification of unstable training regimes. This iterative debugging process was essential in transforming a non-functioning implementation into a stable learning system.

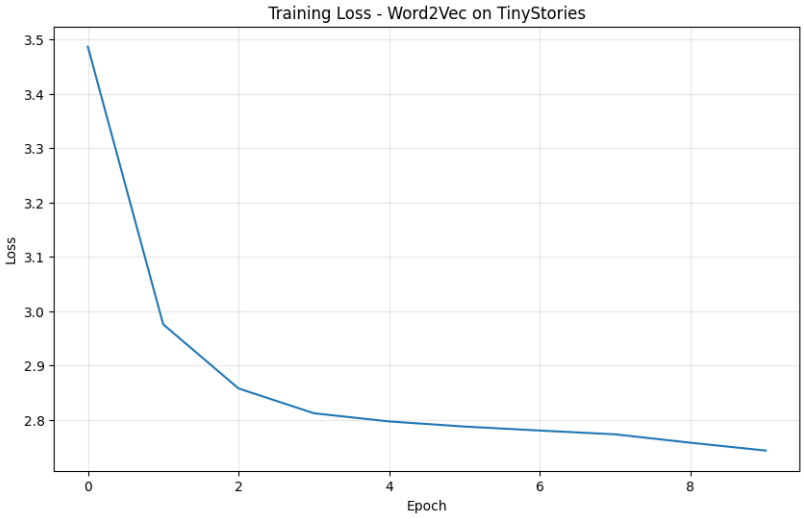

Figure 1

4 Discussion and Possible Improvements

While the implementation is intentionally minimal and framework-free, several practical extensions could further improve both performance and embedding quality. The training procedure could be enhanced with learning rate scheduling, gradient clipping and better weight initialization (such as He or Xavier initialization) to improve convergence stability. On the data side, subsampling of very frequent words and dynamic context window sizes could be introduced to reduce the dominance of common tokens and provide richer training signals. From a modeling and efficiency perspective, alternatives such as hierarchical softmax and mini-batch vectorization could improve scalability to larger corpora. Finally, adding basic evaluation protocols and model checkpointing would make the implementation more useful as a practical research and experimentation baseline.

Overall, this implementation demonstrates a full understanding of the word2vec training procedure at the level of forward computation, loss design, gradient derivation and parameter updates, providing an interpretable baseline for further experimentation.