Implementation and Evaluation of Skip-Gram Word2Vec with Negative Sampling on WikiText-103

Andrew Lisenby
School of Computing
College of Engineering
Wichita State University
ablisenby@shockers.wichita.edu

***Abstract*—This report presents a manual implementation of the Skip-gram Word2Vec model with negative sampling as described by Mikolov et al. (2013). Using the WikiText-103 dataset, we preprocess 80.4 million tokens into a 47,688-word vocabulary, and train a 300-dimensional model for 10 epochs. The final negative-sampling loss reaches 2.6132. Analysis shows semantically coherent nearest neighbors, and analogies solved with pre-trained GloVe 300d vectors match expected results with high cosine values.**

***Index Terms*—Word embeddings, Skip-gram, negative sampling, Word2Vec, GloVe, cosine similarity, word analogies**

I. Introduction

Word2Vec [1] is a two-layer neural network that learns vector representations from an unstructured text corpus. The primary intention is to capture semantic and syntactic relationships in a continuous vector space using high-dimensional embedding vectors, clustering tokens with similar meanings, and enabling meaningful analogies through vector arithmetic. This assignment manually implements the Skip-gram model on the WikiText-103 dataset, evaluates cosine similarity within local neighborhoods, and performs vector arithmetic for analogies with the pre-trained GloVe 300d embeddings [2].

II. Preprocessing

The WikiText-103-raw-v1 training split was loaded via the Hugging Face `datasets` library [3]. The corpus was first converted to lowercase and then tokenized with the regular expression `re.findall(r'\w+', line.lower())`. Only tokens with at least one alpha character, and a length greater than 2 were retained. 80,442,193 tokens were obtained across 1,161,667 sentences.

A vocabulary was constructed using a minimum frequency threshold of **50**. All words appearing fewer than 50 times were replaced with the special `<UNK>` token. This produced a final vocabulary of **47,688** words. The threshold was deliberately set higher through experimentation than typical defaults because WikiText-103 contains only ˜80 million tokens – substantially smaller than the multi-billion-token corpora used in the original Word2Vec work [1]. At a lower threshold, many words would appear too infrequently to be representative of the dataset context.

III. Model Architecture and Training

A Skip-gram model with negative sampling was implemented using two separate embedding tables: $\mathtt{W}_{\text{in}}$ (target word embeddings) and $\mathtt{W}_{\text{out}}$ (context/output word embeddings), both of shape $47{,}688 \times 300$.

The training objective is the negative-sampling loss:

$$\mathcal{L} = -\log \sigma(v_c^\top u_o) - \sum_{k=1}^{10} \log \sigma(-v_c^\top u_k) \tag{1}$$

where v_c is the target word vector, u_o is the true context vector, u_k are 10 negative samples, and σ is the sigmoid function. This training results in a binary classifier to distinguish real context pairs from noise without computing a full softmax over the vocabulary.

Training Process and Design Choices. We initially adopted a large batch size of 8192 combined with a custom `IterableDataset` that generates skip-gram pairs on-the-fly. This design was chosen to maximize GPU utilization, accelerate training throughput, and completely avoid materializing the full set of approximately 804 million training pairs in RAM (which would have required over 50 GB of memory).

However, the large-batch SGD approach led to slow convergence and frequency collapse on the relatively small WikiText-103 corpus. We therefore switched to the Adagrad optimizer (initial LR = 0.05) and reduced the batch size to 2048. These changes provided more frequent gradient updates and adaptive per-parameter learning rates, resulting in smooth convergence with the average negative-sampling loss dropping from 2.81 (epoch 1) to 2.6132 (epoch 10).

After training, we applied the *All-but-the-Top* post-processing technique [4] to reduce anisotropy in the embedding space and improve the reliability of cosine similarity scores.

IV. Experiments and Results

A. Word Similarity

After *All-but-the-Top* post-processing and L2-normalization, we computed the top-10 nearest neighbors for each query word using cosine similarity. Results are shown below:

These neighborhoods reflect the contextual relationships between tokens in the Wikipedia corpus (for example, beverages for *coffee*, or desserts for *cookies*)

TABLE I
TOP-10 MOST SIMILAR WORDS FOR EACH QUERY (COSINE SIMILARITY AFTER ALL-BUT-THE-TOP)

Query	Rank	Similar Word (cosine)
coffee	1	cocoa (0.6194)
	2	tea (0.5745)
	3	beans (0.5392)
	4	teas (0.5274)
	5	drinks (0.5181)
	6	vodka (0.5164)
	7	snacks (0.5151)
	8	cassava (0.5053)
	9	biscuits (0.5016)
	10	starbucks (0.4971)
pasta	1	soups (0.6594)
	2	salad (0.6333)
	3	dishes (0.6297)
	4	desserts (0.6074)
	5	pancakes (0.6052)
	6	sauces (0.6009)
	7	pastry (0.5919)
	8	salads (0.5909)
	9	sauce (0.5860)
	10	falafel (0.5801)
tuna	1	mackerel (0.6588)
	2	shrimp (0.6228)
	3	squid (0.6129)
	4	lobsters (0.6101)
	5	fish (0.6071)
	6	copepods (0.5994)
	7	longline (0.5846)
	8	pelagic (0.5751)
	9	clams (0.5707)
	10	mussels (0.5666)
cookies	1	biscuits (0.6497)
	2	mayonnaise (0.6248)
	3	patties (0.6097)
	4	baked (0.6021)
	5	pancakes (0.5987)
	6	snack (0.5965)
	7	dessert (0.5807)
	8	desserts (0.5798)
	9	snacks (0.5756)
	10	noodles (0.5740)

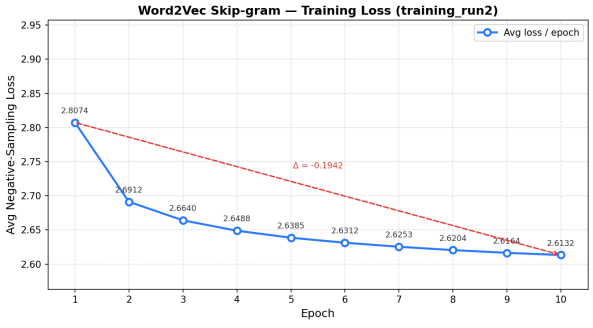

Fig. 1. Training loss curve over 10 epochs (final loss = 2.6132).

B. Word Analogies (GloVe 300d)

Using vector arithmetic $\vec{B} - \vec{A} + \vec{C}$ and cosine ranking with pre-trained GloVe 300d embeddings, the following analogies were solved:

- Spain : Spanish :: Germany : **german** (0.8975)
- Japan : Tokyo :: France : **paris** (0.8097)
- Woman : Man :: Queen : **king** (0.6635)
- Australia : Hotdog :: Italy : **liguria** (0.4466)

These results illustrate that word embeddings encode meaningful linear relationships in vector space. Analogies succeed because the offset created by vector arithmetic (e.g., Spanish − Spain) captures semantic or syntactic patterns that frequently occur in the training corpus. However, some analogies fail when the intended relationship is weak or sparsely represented in the data (as with Australia–Hotdog).

Fig. 2. t-SNE visualization of the top-200 most frequent word embeddings.

V. DISCUSSION AND CHALLENGES

The results confirm that Skip-gram successfully captures distributional semantics even on an unstructured corpus of text such as Wikipedia. Several implementation challenges were encountered and addressed during development:

- **Memory explosion**: Materializing ∼804 million skip-gram pairs would have required over 50 GB of RAM. This was solved by implementing a custom `IterableDataset` that generates training pairs on-the-fly with inline subsampling, keeping memory usage low while maintaining high GPU utilization.
- **Frequency collapse and slow convergence**: The initial large-batch SGD approach (batch size 8192) with linear learning-rate decay failed to capture meaningful semantic qualities. We addressed this by switching to the **Adagrad** optimizer and reducing the batch size to 2048, which provided more frequent updates and adaptive per-parameter learning rates.
- **Anisotropic embeddings**: The learned vectors exhibited the common "narrow cone" problem, leading to spuriously high cosine similarities. We mitigated this using the All-but-the-Top post-processing technique.

No external Word2Vec libraries (such as Gensim or Torchtext) were used at any point; the entire pipeline was implemented from scratch as required by the assignment.

VI. Conclusion

We successfully implemented and evaluated a fully constraint-compliant Skip-gram Word2Vec model. The trained embeddings, vocabulary mappings, and all experimental results are reproducible using the code and artifacts available at https://github.com/alisenby94/word2vec-skipgram.

References

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[2] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proc. Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[3] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[4] J. Mu, S. Bhat, and P. Viswanath, "All-but-the-top: Simple and effective postprocessing for word representations," in *Proc. International Conference on Learning Representations (ICLR)*, 2018.