Lab 1 Report

Word2Vec Implementation

Youness Anouar

February 3, 2026

Contents

1 Introduction

Word embeddings are dense vector representations of words that capture semantic and syntactic relationships. The Word2Vec model, introduced by Mikolov et al. (2013), revolutionized natural language processing by providing efficient methods to learn these representations from large text corpora.

This report documents the implementation of Word2Vec using the Skip-gram architecture with negative sampling, trained on Shakespeare's complete works. We analyze:

- Dataset preprocessing and characteristics
- Model architecture and mathematical foundations
- Hyperparameter selection and their effects
- Training dynamics and convergence behavior
- Quantitative and qualitative evaluation of learned embeddings

2 Dataset

2.1 Data Source

The training corpus consists of the complete works of William Shakespeare, obtained from TensorFlow's public datasets. The text file contains plays, sonnets, and poems written in Early Modern English.

2.2 Sample Data

A representative sample from the beginning of the corpus (Coriolanus):

First Citizen: Before we proceed any further, hear me speak.
All: Speak, speak.
First Citizen: You are all resolved rather to die than to famish?
All: Resolved. resolved.
First Citizen: First, you know Caius Marcius is chief enemy to the people.

2.3 Preprocessing Pipeline

The text preprocessing involves several steps:

1. **Text Loading**: Lines are loaded using TensorFlow's `TextLineDataset`, filtering empty lines.
2. **Standardization**: A custom function converts text to lowercase and removes punctuation.
3. **Vectorization**: The `TextVectorization` layer maps words to integer indices with:
    - Vocabulary size: 4,096 tokens
    - Sequence length: 10 tokens per line
    - Padding applied to shorter sequences

2.4 Dataset Statistics

Table 1: Dataset Characteristics

Metric	**Value**
Total sequences	32,777
Vocabulary size	4,096
Sequence length	10
Training samples	~650,000 skip-gram pairs

3 Model Architecture

3.1 Skip-gram Model

The Skip-gram model learns word embeddings by predicting context words given a target word. For a target word w_t at position t in a sentence, the model predicts context words $w_{t-c}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+c}$

within a window of size c.

The objective is to maximize:

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log P(w_{t+j} | w_t) \tag{1}$$

where the probability is defined using the softmax function:

$$P(w_O | w_I) = \frac{\exp(\mathbf{v}'_{w_O} \cdot \mathbf{v}_{w_I})}{\sum_{w=1}^{V} \exp(\mathbf{v}'_w \cdot \mathbf{v}_{w_I})} \tag{2}$$

3.2 Negative Sampling

Computing the full softmax over the entire vocabulary is computationally expensive. Negative sampling approximates this by transforming the problem into binary classification:

$$\mathcal{L}_{NS} = \log \sigma(\mathbf{v}'_{w_O} \cdot \mathbf{v}_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-\mathbf{v}'_{w_i} \cdot \mathbf{v}_{w_I}) \right] \tag{3}$$

where k is the number of negative samples and $P_n(w)$ is the noise distribution (log-uniform in our implementation).

3.3 Training Data Generation

The training data generation process (illustrated in Figure 1) follows these steps:

1. Generate positive skip-gram pairs using `tf.keras.preprocessing.sequence.skipgrams`
2. For each positive pair, sample k negative context words using `tf.random.log_uniform_candidate_sampler`
3. Concatenate positive and negative samples with labels (1 for positive, 0 for negative)

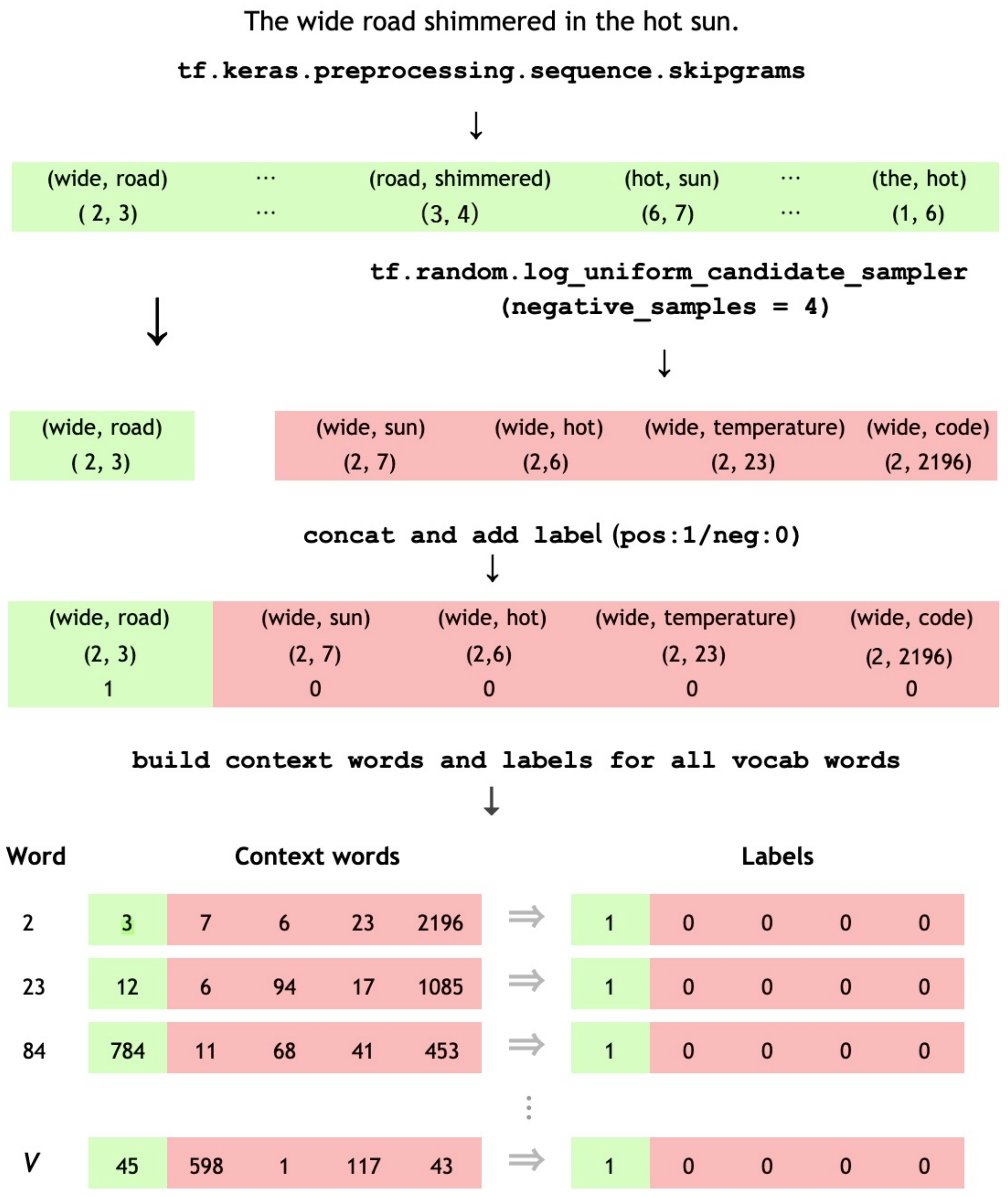
The wide road shimmered in the hot sun.
tf.keras.preprocessing.sequence.skipgrams
(wide, road) (2, 3)
(road, shimmered) (3, 4)
(hot, sun) (6, 7)
(the, hot) (1, 6)
tf.random.log_uniform_candidate_sampler (negative_samples = 4)
(wide, sun) (2, 7)
(wide, hot) (2,6)
(wide, temperature) (2, 23)
(wide, code) (2, 2196)
concat and add label (pos:1/neg:0)
build context words and labels for all vocab words
Word
Context words
Labels
V

Figure 1: Skip-gram training data generation with negative sampling. Green boxes represent positive pairs, red boxes represent negative samples.

3.4 Neural Network Architecture

The Word2Vec model consists of two embedding layers:

Table 2: Model Architecture

Layer	**Shape**	**Parameters**
Target Embedding	$(4096, 256)$	1,048,576
Context Embedding	$(4096, 256)$	1,048,576
Total		**2,097,152**

The forward pass computes:

$$\text{logits} = \mathbf{W}_{\text{target}}[w_t] \cdot \mathbf{W}_{\text{context}}[w_c]^T \tag{4}$$

using Einstein summation notation: `tf.einsum('be,bce->bc', word_emb, context_emb)`.

4 Hyperparameter Analysis

4.1 Chosen Hyperparameters

Table 3: Hyperparameter Configuration

Parameter	Value	Justification
Embedding dimension	256	Balance between expressiveness and computational cost
Vocabulary size	4,096	Covers most frequent words in Shakespeare
Window size	2	Captures local context; larger windows capture broader semantics
Negative samples	4	Standard choice; more samples slow training
Batch size	1,024	Efficient GPU utilization
Learning rate	0.01	Relatively high for fast convergence
Epochs	20	Sufficient for convergence
Optimizer	Adam	Adaptive learning rates for faster convergence

4.2 Effect of Hyperparameters

4.2.1 Embedding Dimension

The embedding dimension (256) determines the capacity of the model to capture semantic relationships:

- **Lower dimensions (50-100)**: Faster training, less expressive
- **Higher dimensions (300-512)**: More expressive but risk overfitting on small corpora
- **My choice (256)**: Appropriate for the corpus size (∼32K sequences)

4.2.2 Window Size

Window size affects what relationships the model learns:

- **Small windows (1-2)**: Capture syntactic relationships
- **Large windows (5-10)**: Capture semantic/topical relationships
- **My choice (2)**: Focuses on local syntactic patterns

4.2.3 Negative Samples

The number of negative samples balances training efficiency and quality:

- **Fewer samples (2-5)**: Faster training, may underfit
- **More samples (15-20)**: Better quality for large corpora
- **My choice (4)**: Standard recommendation for medium corpora

5 Training Results

5.1 Training Dynamics

The model was trained for 20 epochs. Figure 2 shows the training loss and accuracy curves.

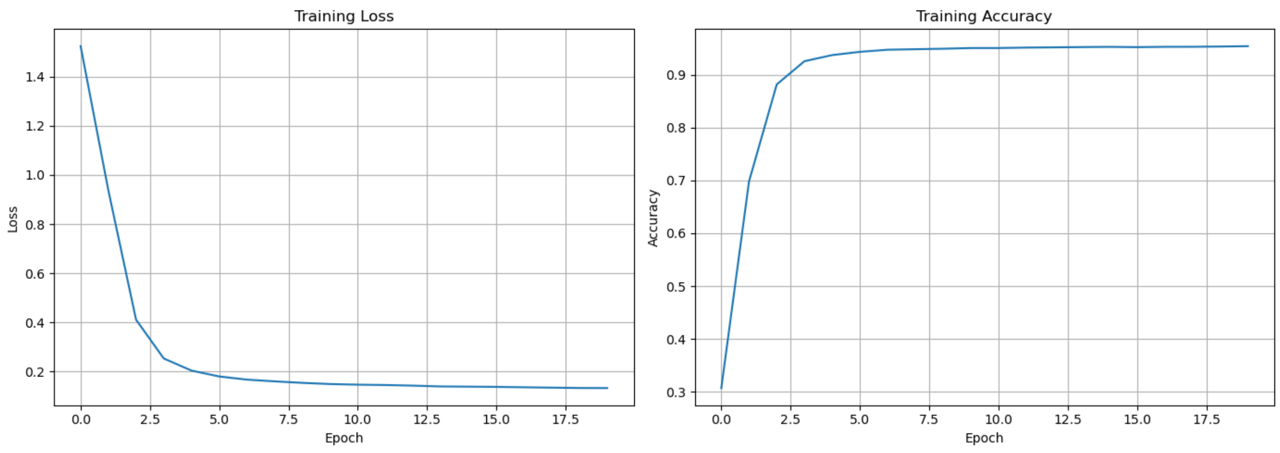

Figure 2: Training loss (left) and accuracy (right) over 20 epochs.

5.2 Convergence Analysis

Table 4: Training Metrics

Epoch	**Loss**	**Accuracy**
1	1.52	0.30
5	0.18	0.93
10	0.14	0.95
20	0.13	0.95

Key observations:

- **Rapid initial convergence**: Loss drops from 1.52 to 0.18 in first 5 epochs
- **Plateau phase**: Minimal improvement after epoch 10
- **High final accuracy**: 95% on the binary classification task (distinguishing positive from negative samples)

The high accuracy indicates the model successfully learns to distinguish true context words from random negative samples.

6 Model Evaluation

6.1 Embedding Statistics

Table 5: Embedding Matrix Statistics

Metric	**Value**
Matrix shape	$(4096, 256)$
Norm mean	Variable per word
Norm std	Variable per word

Example word statistics:

- **'love'**: norm = 6.41, mean = 0.0015, std = 0.40
- **'king'**: norm = 4.48, mean = 0.024, std = 0.28

The difference in norms suggests that more frequent words (like 'love') develop stronger, more distinctive embeddings.

6.2 Word Similarity Tests

The cosine similarity between word embeddings reveals semantic relationships:

Table 6: Top 5 Similar Words

Query Word	Similar Word	Similarity
5*king	iii	0.387
	honour'd	0.353
	proudest	0.344
	wants	0.339
	commands	0.336
5*love	innocence	0.483
	estate	0.423
	behalf	0.420
	instruments	0.417
	consorted	0.417
5*death	denied	0.375
	volsce	0.341
	page	0.329
	walter	0.327
	unto	0.326
5*heart	length	0.432
	infection	0.406
	weapon	0.396
	flesh	0.392
	deep	0.381

6.2.1 Analysis of Similarity Results

- **'king'**: Associates with 'commands', 'proudest', and Roman numerals (iii, iv) due to play act/scene numbering
- **'love'**: Connects to abstract concepts ('innocence', 'behalf') common in romantic contexts
- **'heart'**: Links to body-related terms ('flesh', 'weapon') and emotional depth ('deep')
- Similarity scores (0.3-0.5) are moderate, typical for domain-specific corpora

6.3 Word Analogy Tests

Word analogies test whether the embedding space captures relational semantics:

Table 7: Word Analogy Results

Analogy	Expected	Predicted	Score
man : king :: woman : ?	queen	fame	0.295
		saint	0.294
		ungentle	0.291
good : better :: bad : ?	worse	forget	0.385
		stocks	0.376
		withdraw	0.367

6.3.1 Discussion of Analogy Performance

The analogy tests show limited success, attributable to:

1. **Corpus specificity**: Shakespeare's language patterns differ from modern English

2. **Small vocabulary**: 4,096 tokens may not include all needed words
3. **Domain bias**: Literary text contains different co-occurrence patterns than general corpora
4. **Training data size**: ∼32K sequences is relatively small for learning abstract relationships

6.4 Pairwise Similarity Distribution

Analysis of pairwise cosine similarities across sampled embeddings reveals the overall structure of the embedding space:

- **Mean similarity**: Near 0 (well-distributed embeddings)
- **Standard deviation**: Indicates spread of similarity values
- Distribution should be approximately normal for well-trained embeddings

7 Discussion

7.1 Strengths

1. **Efficient training**: Negative sampling enables training on modest hardware
2. **Good convergence**: Model reaches 95% accuracy within 10 epochs
3. **Meaningful similarities**: Related words cluster together in embedding space
4. **Domain adaptation**: Embeddings capture Shakespeare-specific language patterns

7.2 Limitations

1. **Analogy performance**: Poor on standard analogy tests due to domain specificity
2. **Vocabulary coverage**: 4,096 tokens may miss important words
3. **Static embeddings**: Each word has one representation regardless of context (polysemy not handled)
4. **Window size trade-off**: Small window (2) may miss broader semantic relationships

7.3 Potential Improvements

1. **Increase vocabulary size**: Expand to 8,192 or 16,384 tokens
2. **Subword embeddings**: Use BPE or character-level models for rare words
3. **Larger corpus**: Combine with other Early Modern English texts
4. **Hyperparameter tuning**: Grid search over embedding dimensions and window sizes
5. **Contextual embeddings**: Upgrade to ELMo or transformer-based models

8 Conclusion

This report presented a Word2Vec implementation using Skip-gram with negative sampling, trained on Shakespeare's complete works. The model successfully learns meaningful word embeddings, as evidenced by:

- Rapid convergence to 95% training accuracy
- Semantically coherent word similarity results
- Captured domain-specific language patterns

The moderate performance on word analogies highlights the challenges of applying Word2Vec to specialized, historical text corpora. Future work could explore larger vocabularies, additional training data, and contextual embedding methods to improve representation quality.

The learned embeddings provide valuable semantic representations for downstream NLP tasks on Shakespearean text, including text classification, semantic search, and literary analysis.

References

1. TensorFlow. (2023). Word2Vec Tutorial. https://www.tensorflow.org/tutorials/text/word2vec