Sentiment Analysis of Movie Reviews

Classifying movie reviews as positive or negative

January 2026

SUBMITTED TO
University of Craiova
Faculty of Automation, Computers and Electronics
Prof. dr. ing. Costin Bădică

SUBMITTED BY
Dan-Cosmin Dăgădiță
dagadita.dan.v7d@student.ucv.ro

Contents

1. Introduction

Sentiment analysis is the science of teaching computers how to understand human emotions within text. Instead of a human having to read thousands of movie reviews to see if a film was well-received, we use Natural Language Processing to automatically label them as "Positive" or "Negative." This is useful for businesses and researchers because it allows them to process unstructured data-like social media comments or customer feedback [1].

> **Definition 1**
>
> **Natural Language Processing (NLP):** A field of Artificial Intelligence that focuses on the interaction between computers and human language. Its goal is to enable computers to understand, interpret, and generate text in a way that is valuable [1].

However, understanding human language is a difficult task for a machine. The challenge is in mostly these two areas:

- **Language Nuances:** Humans often use sarcasm, complex descriptions, or words that change meaning depending on the context.
- **Long-Term Dependencies:** In a long movie review, a critic might start with a specific detail and not reveal their final opinion until the very end. A computer needs to "remember" the beginning of the story to understand the conclusion.

To solve these problems, this report uses **Deep Learning**, specifically **Recurrent Neural Networks (RNNs)**. Unlike traditional programs that look at words in isolation, an RNN processes text as a sequence. It reads one word at a time while maintaining an internal "memory" of what it has already seen [1].

In this project, we specifically focus on **Long Short-Term Memory (LSTM)** and **Gated Recurrent Units (GRU)**. These are "smarter" versions of the standard RNN. They were created to fix a specific technical flaw called the **vanishing gradient**, which often causes basic models to "zone out" or lose their way during long sentences [2].

> **Definition 2**
>
> **Vanishing Gradient:** Think of this as a "fading signal." When a model is trying to learn from a very long sentence, the signal it uses to update its weights gets weaker and weaker as it travels back to the start of the sentence. Eventually, the signal disappears (vanishes), and the model stops learning [2].

2. Deep Learning Methodology

The methodology used in this project follows a **sequence-to-vector** architecture [1]. The process is broken down into four main steps:

- **Text Vectorization:** This is the first step where the computer turns sentences into a list of ID numbers. Since computers don't understand letters, we give the 10,000 most common words their own "ID number" [1].

- **Word Embeddings:** After we have the ID numbers, we turn them into **dense vectors**. Think of this as giving every word its own set of coordinates on a map of "meaning" [1]. Pre-trained word embeddings for this project were sourced from Stanford [3] and Kaggle [4].

Definition 3

Word Embeddings: A technique where words are represented as vectors of numbers. This allows the computer to calculate the "distance" between words. For example, "king" and "queen" would be mathematically closer to each other than "king" and "bicycle" [1].

- **Recurrent Layers:** This is the "brain" of the model that reads the review word by word [1].
  - **SimpleRNN:** The most basic version; it remembers the previous word.
  - **LSTM and GRU:** Advanced versions that have a "long-term memory" [2].
- **Bidirectionality:** Normally, a model reads from left to right. By making it **Bidirectional**, we let the model read the review in both directions simultaneously to capture full context [2].

3. Software Design and Implementation

The software for this project was built using **TensorFlow 2** and **Keras**. Instead of writing six separate programs, we created one flexible function called `build_model`. This acted like a **model factory**: we could tell it exactly what kind of "brain" to use and whether to use pretrained GloVe embeddings or start from scratch.

To keep the training process fast and smooth, we used a specialized **Data Pipeline** (`tf.data.Dataset`).

- **Batching:** We grouped data into sets of 128.
- **Prefetching:** A trick where the computer prepares the **next** batch of data while the model is still processing the current one.

One of the biggest risks in AI is **overfitting**. To prevent this, we used **dropout layers**.

Definition 4

Overfitting: A mistake where a model learns the training data **too** well, including its random noise and specific examples. This makes the model "stiff"-it performs perfectly on the training data but fails on new, real-world data because it hasn't learned the general pattern.

Definition 5

Dropout: A regularization technique where we randomly "ignore" some neurons during training. This prevents the model from becoming too reliant on specific words and forces it to learn more robust patterns.

Infrastructure and Reproducibility

The project is designed for portability using Docker Compose and a Google Colab Jupyter environment. The local runtime setup utilizes a dedicated container to handle the Nvidia GPU environment via the Nvidia Container Toolkit. Data and embeddings are managed in a local `files` directory, ensuring the project remains independent of specific cloud-only features. This allows the notebooks to be executed consistently across local IDEs like VS Code and the hosted Google Colab environment.

4. Dataset

For this project, we used the **IMDB Movie Reviews Dataset** [5]. The dataset contains **50,000 reviews**. These are described as "highly polar," meaning they are either clearly positive or clearly negative [1].

To make sure the model actually learns, we split the data into two groups:

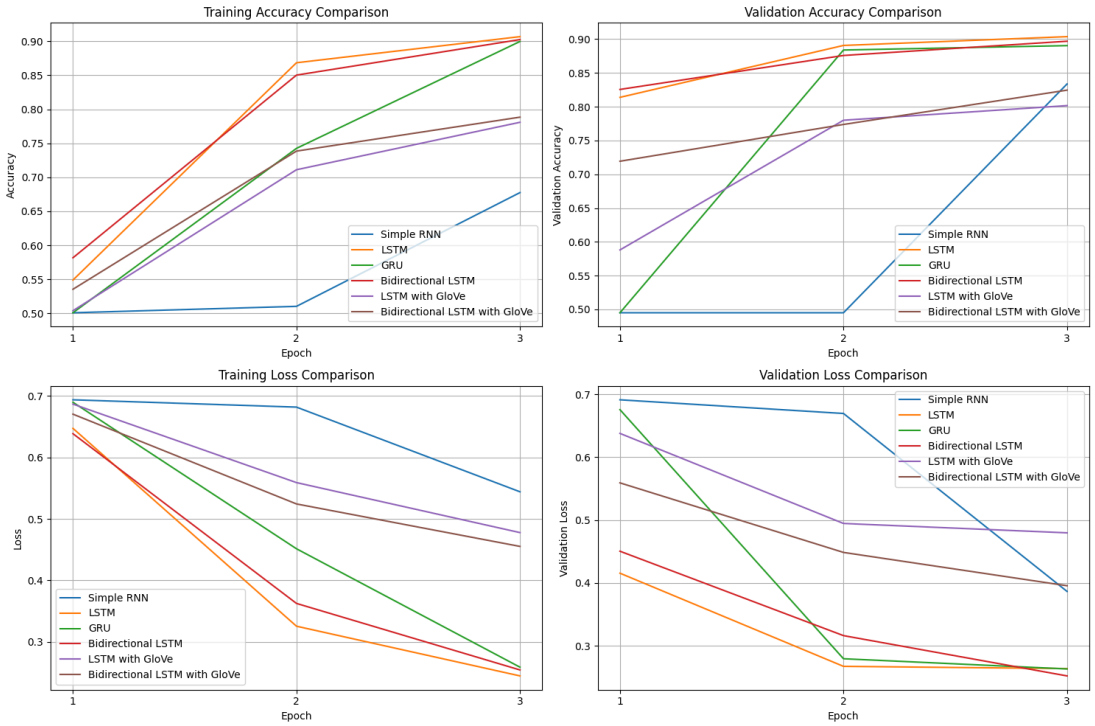
- **Training Set (40,000 reviews):** The "textbook" used to learn.
- **Testing Set (10,000 reviews):** The "final exam" with reviews the model has never seen before.

When the model "studies," we use a process called **batching**.

> **Definition 6**
>
> **Batch Size:** The number of training examples processed in one pass. A batch size of 128 means the model looks at 128 reviews, calculates its error, updates its weights, and then moves to the next 128.

5. Experiments and Results

After training the six different models for 3 rounds (epochs), we compared their performance.



Definition 7

Epoch: One complete pass of the entire training dataset through the neural network [1]. Training for multiple epochs allows the model to refine its "guesswork" and become more accurate.

The **standard LSTM** was the top performer with **89.94% accuracy**. Even though it is simpler than the Bidirectional version, it was the most efficient [2].

- The **GRU (89.31%)** and **Bidirectional LSTM (89.45%)** were very close behind the winner [2].
- The **Simple RNN (83.71%)** had the lowest accuracy because of its "short-term memory" issues [1].

The **GloVe** models were expected to be the strongest but performed the worst (around **80-82%**). This is likely because GloVe was trained on general Wikipedia facts, whereas movie reviews use very specific, emotional language that the "random" models learned better from scratch.

6. Conclusion

The experiments show that the **standard LSTM** was the most successful model (89.94%). It performed better than the **Simple RNN** because it can remember important words even in very long reviews [2].

A surprising result was that **randomly initialized embeddings** performed better than **pretrained GloVe embeddings**. This is likely because the random embeddings were

allowed to learn the specific “slang” of movie critics, whereas GloVe was restricted by its general-purpose training.

Additionally, making the models more complex (Bidirectional) did not help much. This suggests that for a simple “Positive or Negative” task, a standard LSTM is already powerful enough [2].

References

[1] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. Online, 2023. [Online]. Available: https://d2l.ai/chapter_recurrent-neural-networks/index.html

[2] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. Online, 2023. [Online]. Available: https://d2l.ai/chapter_recurrent-modern/index.html

[3] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation." [Online]. Available: https://nlp.stanford.edu/projects/glove/

[4] D. George, "GloVe 6B 100d." [Online]. Available: https://www.kaggle.com/datasets/danielwillgeorge/glove6b100dtxt

[5] N. Lakshmi, "IMDB Dataset of 50K Movie Reviews." [Online]. Available: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews