A Report on Binary Sentiment Analysis using Traditional and Transformer-Based Embeddings and Classification Neural Network

By: Vedant Kanawade

1. INTRODUCTION

1.1 Background

Sentiment Analysis, also known as *Opinion Mining*, is a subfield of Natural Language Processing (NLP) that focuses on identifying and classifying the polarity of opinions or emotions expressed in text. The task aims to determine whether a given piece of text conveys a positive or negative sentiment or in a variety of range of emotions.

It plays a crucial role in understanding user opinions across various domains such as product reviews, social media posts, and customer feedback. Businesses use sentiment analysis to gauge public perception of their products, monitor customer satisfaction, and make data-driven decisions. With the rapid growth of online platforms, automated sentiment classification has become increasingly important to process vast amounts of unstructured textual data efficiently.

1.2 Problem Definition

The primary goal of this project is to classify product reviews into positive or negative sentiment categories.
The dataset provided consists of two files:

- train.json – Contains reviews with corresponding sentiment labels (0 = negative, 1 = positive).
- test.json – Contains reviews without labels, used for prediction and evaluation.

The task is framed as a supervised binary classification problem, where models learn from labeled examples in the training data and predict the sentiment of unseen reviews in the test set. The input data is in raw text form, which must be transformed into numerical feature representations before feeding into a classifier.

1.3 Objective

The main objective of this project is to build a learning-based sentiment classifier that leverages advanced text embeddings combined with a neural network classifier. Specifically, we will explore two types of embeddings: traditional TF-IDF embeddings derived from a bag-of-words (BoW) model, and contextual embeddings generated using Sentence-BERT (SBERT) via the Transformers library. These embeddings will be used as input to a small neural network classifier, which consists of one or more fully connected layers acting as the classification head to produce sentiment predictions.

The project aims to:

- Experiment with model parameters (e.g., learning rate, embedding dimension, batch size) to achieve optimal performance.
- Compare the performance of classifiers trained with TF-IDF embeddings versus SBERT embeddings, and select the best combination of embedding and classifier.
- Evaluate the selected classifier on the test set.
- Analyze both correctly and incorrectly classified samples to identify strengths and weaknesses of the model.

2. LITERATURE SURVEY

2.1 Problem Definition

Sentiment analysis, also known as opinion mining, is a field of study within Natural Language Processing (NLP) that uses computational techniques to systematically identify, extract, and quantify the subjective information in text. Its core goal is to determine the emotional tone or attitude of a piece of writing, such as a review, tweet, or survey response, classifying it as positive, negative, or neutral.

Given a piece of text (sentence, paragraph, or document), the task is to classify its sentiment into predefined categories (e.g., positive, negative, neutral). Key problem variations include:

1. **Supervised vs. Unsupervised:**
    a. **Supervised:** Requires labeled data; models learn directly from examples. Most state-of-the-art methods are supervised.
    b. **Unsupervised:** Uses lexicons, clustering, or pattern-based heuristics to infer sentiment without labeled data. Useful when labeled data is scarce.
    c. **Challenges:**
        - Supervised: Requires large labeled datasets.
        - Unsupervised: May be less accurate; depends on lexicons or clustering heuristics.
    d. **Common Solutions:**
        - Supervised: Use labeled datasets with classifiers (Logistic Regression, SVM, NN).
        - Unsupervised: Lexicon-based approaches, clustering, or rule-based methods.

2. **Closed-set vs. Open-set:**
    a. **Closed-set:** Sentiment categories are predefined.
    b. **Open-set:** Models may encounter unseen or emerging sentiment categories, requiring mechanisms to handle unknown labels or uncertainty.
    c. **Challenges:**
        - Open-set: Unseen sentiment classes, requiring uncertainty handling.
        - Closed-set: Limited flexibility for emerging sentiment categories.
    d. **Common Solutions:**
        - Closed-set: Standard classification models.
        - Open-set: Out-of-distribution detection, thresholding on prediction confidence, incremental learning.

3. **Domain Shift / Cross-domain Sentiment Analysis:**
    a. Domain shift refers to the situation where the statistical properties of the data in the training domain differ from those in the target domain. In sentiment analysis, this means that a model trained on one type of text (e.g., movie reviews) may not perform well on another type of text (e.g., product reviews)
    b. **Challenges:**
    - Models trained in one domain may fail in another.
    - Vocabulary differences and context variations across domains.
    c. **Common Solutions:**

- Fine-tuning pre-trained models on target domain.
- Transfer learning, domain adaptation, multi-domain embeddings.

4. **Fine-grained vs. Coarse-grained Sentiment:**
    a. **Coarse-grained:** Classifies overall sentiment of a text (positive, negative, neutral).
    b. **Fine-grained:** Detects sentiment at a more granular level, e.g., sentence-level, aspect-level (like product features or service attributes).
    c. **Challenges:**
        - Detecting fine-grained sentiment for specific aspects is complex.
        - Mixed sentiments within a text.
    d. **Common Solutions:**
        - Aspect-based sentiment analysis using attention mechanisms.
        - Multi-task learning combining sentiment classification and aspect detection.

5. **Multilingual vs. Monolingual Sentiment Analysis:**
    a. Many applications require handling multiple languages, like language translation.
    b. **Challenges:**
        - Lack of labeled data for low-resource languages.
        - Translation ambiguity or code-mixing.
    c. **Common Solutions:**
        - Multilingual transformers (mBERT, XLM-R).
        - Cross-lingual transfer learning and zero-shot classification.

6. **Real-time / Streaming Sentiment Analysis:**
    a. Applications like social media monitoring require sentiment prediction in real time.
    b. **Challenges:**
        - Latency constraints; large models slow inference.
        - Continuous input stream requires incremental updates.

    Common Solutions:
        - Lightweight embeddings (SBERT, DistilBERT).
        - Online learning or incremental update models.

2.2 Approaches to Sentiment Analysis

Sentiment analysis systems use a combination of methods, often leveraging Natural Language Processing (NLP) and Machine Learning (ML):

1. **Rule-Based Approaches:** This method relies on pre-defined dictionaries, or lexicons, of words that are manually scored for sentiment.

    - For example, words like "fantastic" might be scored +0.8, while "awful" might be scored -0.9. The system tallies up the scores of all words in a sentence, adjusting for intensifiers ("very") and negators ("not"), to calculate a final sentiment score.

2. **Machine Learning (ML) Approaches:** Algorithms (like Naïve Bayes, Support Vector Machines, or Deep Learning models) are trained on vast amounts of text that has already been labeled as positive, negative, or neutral.

    - The model learns to associate patterns of words, word order, and context with the corresponding sentiment label, allowing it to predict the sentiment of new, unseen text.

3. **Hybrid Approaches**: These systems combine the interpretability of rule-based systems with the accuracy of machine learning models.

2.3 Developments in Natural Language Processing (NLP)

2.3.1 Early NLP and Sentiment Analysis

Natural Language Processing (NLP) began as a rule-based and symbolic field, where linguists manually defined grammar rules and lexicons for tasks such as part-of-speech tagging, parsing, and text classification. Early sentiment analysis relied heavily on lexicon-based approaches, which assign sentiment scores to words and aggregate them to classify text sentiment.

- The paper, "Thumbs Up? Sentiment Classification Using Machine Learning Techniques," published in 2002, demonstrated that supervised machine learning methods such as Naive Bayes, Maximum Entropy, and SVM outperform lexicon-based methods for movie review sentiment classification.

- Similarly, the paper "Thumbs Up or Thumbs Down?" introduced the concept of using pointwise mutual information (PMI) with web search engines to derive semantic orientation for words, marking one of the first unsupervised sentiment analysis attempts.

These methods were limited by vocabulary coverage and inability to capture context or polysemy, motivating the shift toward distributed representations.

2.3.2 Word Embeddings and Neural Approaches

The introduction of word embeddings revolutionized NLP by providing dense, low-dimensional vector representations capturing semantic similarity.

- Word2Vec introduced continuous bag-of-words (CBOW) and skip-gram models, which became widely adopted in NLP pipelines.

- GloVe combined global co-occurrence statistics with neural embeddings to produce better semantic representations.

For sentiment analysis, embeddings allowed neural networks, such as feedforward networks, CNNs and LSTMs to capture semantic and syntactic patterns more effectively than sparse bag-of-words representations.

- The research titled, “Convolutional Neural Networks for Sentence Classification” showed that CNNs applied to word embeddings could extract n-gram features for sentence-level classification.

- The paper, “Document Modeling with Gated Recurrent Neural Network” used gated RNNs to capture sequential dependencies, improving sentiment detection in longer texts.

Limitations: Context-independent embeddings cannot differentiate word senses in different contexts, and sentence-level semantics are still inadequately captured.

2.3.3 Contextualized Embeddings and Transformers

The advent of transformer architectures marked a turning point in NLP. Models such as BERT, RoBERTa, and SBERT provide contextual embeddings, capturing nuanced sentence-level semantics.

- The famous research paper “Attention is All You Need” published in 2017 introduced the transformer architecture, replacing recurrence with self-attention mechanisms for efficient context modeling.
- Following up, BERT showed that pre-trained bidirectional transformers could be fine-tuned for downstream NLP tasks, including sentiment classification, achieving state-of-the-art results.
- In 2019, SBERT adapted BERT to produce fixed-size sentence embeddings suitable for semantic similarity and classification tasks, making it efficient for sentiment analysis without fine-tuning the full BERT model.
- Further, Universal Sentence Encoder demonstrated lightweight sentence embeddings for transfer learning in NLP tasks, including sentiment classification.

Contextual embeddings outperform traditional word embeddings in capturing polysemy, sarcasm, and sentence-level semantics, making them the preferred choice for modern sentiment classifiers.

2.3.4 Domain Adaptation and Cross-domain Sentiment Analysis

Recent research addresses domain shift, where models trained on one dataset fail to generalize to another due to differences in vocabulary or style.

- In the paper, “Domain Adaptation for Large-Scale Sentiment Classification” researchers used stacked autoencoders for cross-domain adaptation.
- The research titled “Deep Unsupervised Domain Adaptation” applied adversarial training to align source and target feature distributions.

Transformers with fine-tuning or multi-domain pre-training (e.g., mBERT, XLM-R) provide robust embeddings for cross-domain sentiment analysis.

2.3.5 Aspect-based and Fine-grained Sentiment Analysis

Beyond coarse-grained classification, aspect-based sentiment analysis (ABSA) identifies sentiment towards specific entities or attributes:

- SemEval ABSA Task standardized datasets and evaluation for aspect-level sentiment.
- The paper, "BERT Post-training for Aspect-based Sentiment Analysis" showed that fine-tuning BERT on aspect-specific corpora significantly improves classification performance.

Transformers allow ABSA tasks to capture both global sentence sentiment and localized aspect sentiment.

2.3.6 Recent Trends and Popular Methods

- Lightweight Transformers: DistilBERT, TinyBERT, and SBERT for fast inference in real-time applications.
- Hybrid Approaches: Combining traditional features (TF-IDF, BoW) with contextual embeddings to improve performance in low-resource settings.
- Multi-task Learning: Jointly learning sentiment classification and related tasks such as emotion detection, sarcasm detection, or intent classification.
- Multilingual Models: mBERT, XLM-R, and cross-lingual transfer learning to handle multiple languages or code-mixed data.

2.4 Recent Progress in Sentiment Analysis and NLP

Over the past decade, sentiment analysis has evolved dramatically thanks to advances in representation learning, deep neural networks, and transfer-learning from large pretrained models. Early systems relied on handcrafted lexicons and rule-based techniques, but modern approaches leverage dense embeddings (word-level and sentence-level), large-scale pre-training, and fine-tuning on domain-specific tasks. The field is now characterized by:

- Embeddings that capture contextual meaning rather than simply counting word occurrences.
- Models trained across tasks and domains, enabling transferability and domain adaptation.
- Approaches that handle real-world complications: domain shift, multilingual data, aspect-level sentiment, and real-time deployment.
- Open-source tools and models (e.g., Sentence-BERT, Hugging Face libraries) that democratize research and application.

Therefore, identifying research groups that consistently contribute high-impact work, both in foundational models and in applied NLP, is useful for understanding where the field is headed. Below are two such groups that stand out.

2.4.1 Google Research – Language/Brain/Natural Language Understanding Teams

Research Focus:
Google's NLP research teams focus on building models and systems capable of understanding and generating natural language at scale. Their work spans language representation learning, multilingual modeling, summarization, classification (including sentiment), and grounded language understanding (text + other modalities). They emphasize scalability, cross-lingual support, and practical deployment in products (e.g., search, assistant).

Major contributions to date:

- The release of the model BERT ("Bidirectional Encoder Representations from Transformers") in 2018/2019, which became a foundation for almost all downstream NLP tasks including sentiment analysis.
- Work on large-scale contextual embeddings, multilingual models, and representation learning for text classification, including sentiment and other affective tasks.
- Creating and open-sourcing tools, datasets, and frameworks used by the broader research community, thereby accelerating innovation.

Planned or emerging directions:

- Expanding language and domain coverage: the team is investing in under-represented languages and low-resource settings (e.g., African languages).
- Efficiency and deployment: models that run at scale across Google's products, optimizing inference, latency, and adaptivity.
- Responsible and robust NLP: safe, fair, multilingual, interpretable models across domains.
- Multimodal and grounded understanding: combining text with vision, speech, structured knowledge, and reasoning.

As our project aims to build a sentiment classifier using transformer based embeddings + a small neural network, the Google team's work provides the theoretical backbone (state-of-the-art embedding + fine-tuning paradigm).

2.4.2 UKP Lab (TU Darmstadt)

Research Focus:
The UKP Lab at the Technische Universität Darmstadt (TU Darmstadt) specialises in Natural Language Processing with emphasis on representation learning, large language models, conversational AI, question answering, cross-document processing, and applications across society (e.g., mental health).

Major contributions to date:

- They developed the library and model family Sentence-BERT (SBERT) which repurposes BERT to generate efficient fixed-size sentence embeddings suited for similarity search, clustering and classification.

- They have released extensive open-source resources and frameworks (datasets, embedding models, software) widely adopted in the community. For example, the transition of Sentence Transformers to the Hugging Face ecosystem shows broad impact.
- They engage in interdisciplinary and applied research: e.g., NLP for mental health, AI for science, conversational systems grounded in literature.

Planned or emerging directions:

- Scaling sentence embedding technology further for real-world applications (e.g., retrieval, clustering, domain adaptation) and transferring research to production ecosystems like Hugging Face.
- Focus on privacy-aware AI and domain-sensitive NLP: e.g., models for healthcare, multimodal data, human–AI collaboration.
- Research spanning long documents, cross-document relations, conversational AI grounded in scientific literature and effective domain transfer.

2.5 Major Research Milestones Driving Breakthroughs in NLP

The evolution of Natural Language Processing has been shaped by a series of groundbreaking models that redefined how machines understand and process human language. Among these, the introduction of the Transformer architecture and the development of Sentence-BERT (SBERT) stand out as seminal contributions. Transformers established the foundation for efficient, context-aware sequence modelling, while SBERT adapted these contextual embeddings to produce fixed-size, semantically meaningful sentence representations, enabling fast and scalable downstream applications such as sentiment analysis, semantic search, and clustering. Together, these models illustrate the shift from traditional word-level representations to powerful, sentence-level embeddings that capture nuanced meaning and facilitate real-world NLP applications. Below is the in-depth summary of the two research papers.

2.5.1 "Attention Is All You Need" (Vaswani et al., 2017)

Conference: NeurIPS 2017
Authors: Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Łukasz Kaiser, Illia Polosukhin
Institution: Google Brain & Google Research

I. Overview

This paper revolutionized Natural Language Processing (NLP) by introducing the Transformer architecture, which removed the need for recurrence (RNNs) or convolutions (CNNs) in sequence modeling. It showed that self-attention alone could model dependencies between tokens efficiently, hence the title *"Attention Is All You Need."* The Transformer became the foundation for almost every modern NLP model, including BERT, GPT, T5, SBERT, and many others.

II. Motivation

Before Transformers, most models used Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) architectures for sequential data. However, these had two major drawbacks:

1. **Sequential computation**: tokens were processed one at a time, making training slow and difficult to parallelize.
2. **Long-range dependencies**: RNNs struggled to capture relationships between distant words due to vanishing gradients.

Transformers replaced recurrence with attention mechanisms, enabling:

- Full parallelization during training.
- Better modeling of global context.

III. Architecture Overview

The Transformer consists of an Encoder and a Decoder; each built from stacked layers of self-attention and feedforward networks.

1. Encoder

- Contains *N* identical layers.
- Each layer has:
  - **Multi-Head Self-Attention:** Allows each word to attend to all other words in the input sequence simultaneously.
  - **Feedforward Network:** Applies nonlinear transformations independently to each token's representation.
- Residual connections and Layer Normalization stabilize training.

2. Decoder

- Also has *N* layers.
- Includes:
  - **Masked Self-Attention:** Prevents the model from attending to future positions during training (crucial for autoregressive tasks like translation).
  - **Encoder-Decoder Attention:** Lets the decoder attend to the encoder's output, connecting input and output sequences.

IV. Core Mechanism: Scaled Dot-Product Attention

Given queries Q, keys K, and values V:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

This computes a weighted combination of values, where weights are proportional to similarity between query and key vectors.

Multi-Head Attention: Instead of a single attention function, multiple heads run in parallel to capture different types of relationships between words (e.g., syntactic vs semantic).
Outputs are concatenated and linearly projected.

Positional Encoding: Since the model has no recurrence, it uses positional encodings to inject order information. These are added to input embeddings to help the model understand token positions in the sequence.

V. Key Results

- Achieved state-of-the-art results on WMT 2014 English–German and English–French translation tasks, with far less training time than RNNs.
- Enabled full GPU parallelization, drastically reducing training cost.
- Set the foundation for pretraining + fine-tuning frameworks that dominate modern NLP.

VI. Impact

- The Transformer architecture became the **universal backbone** for NLP (and later, Vision Transformers, Audio Transformers, etc.).
- All major models — **BERT, GPT, T5, RoBERTa, SBERT, DeBERTa, etc.** — are extensions or adaptations of this paper's ideas.

2.5.2 "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks"

Conference: EMNLP 2019
Authors: Nils Reimers, Iryna Gurevych
Institution: UKP Lab, TU Darmstadt

I. Motivation

BERT produces powerful contextual embeddings, but it's computationally expensive for sentence-level similarity tasks. To compare two sentences, one must feed both into BERT together and compute attention across the pair this scales quadratically with dataset size, making it infeasible for tasks like:

- Semantic textual similarity (STS)
- Sentence clustering
- Information retrieval

Sentence-BERT (SBERT) addresses this limitation by generating fixed-size sentence embeddings that can be compared efficiently using cosine similarity.

II. Architecture

SBERT modifies the BERT model by adding a **Siamese or Triplet network structure**:

- Two (or three) BERT encoders share weights.
- Each sentence is independently passed through BERT.
- The final embedding is obtained by mean pooling over the token embeddings (or using the [CLS] token).

This structure enables direct comparison between sentence embeddings without recomputing pairwise BERT interactions.

III. Training Objectives

SBERT fine-tunes BERT on **sentence-pair datasets** using one of several loss functions:

1. **Classification Objective (Natural Language Inference – NLI)**
    - Datasets: SNLI, MultiNLI.
    - Model learns to encode semantic relationships (entailment, contradiction, neutral).
    - Sentence embeddings cluster based on semantic meaning.
2. **Regression Objective (Semantic Textual Similarity – STS)**
    - The cosine similarity between embeddings is trained to match human-annotated similarity scores.
3. **Triplet Loss Objective**
    - Pulls semantically similar sentences closer and pushes dissimilar ones apart.

IV. Advantages

- **Efficiency:** After encoding, sentence comparisons reduce to fast cosine similarity.
- **Scalability:** Enables retrieval, clustering, and semantic search over millions of sentences.
- **Performance:** Achieves significant improvement over traditional BERT in STS benchmarks.

V. Results

- On STS Benchmark, SBERT improved Spearman correlation from 0.77 (BERT baseline) to 0.84.
- Enabled semantic search tasks that are 100 times faster than BERT pairwise encoding.
- Public models like all-MiniLM-L6-v2 (distilled SBERT) are widely used in industry.

VI. Impact

- SBERT bridged the gap between contextual embeddings and efficient semantic similarity computation.
- It became the standard for generating semantically meaningful sentence embeddings in many applications including sentiment classification, information retrieval, and clustering.
- Many modern text representation systems (e.g., SimCSE, LaBSE, OpenAI embeddings) are built upon SBERT's principles.

2.5 Baseline Selection and Proposed Approach

Based on the comprehensive survey of sentiment analysis methods, a hybrid approach combining embeddings with a neural network classifier emerges as the most suitable baseline for this project. Specifically, two embedding strategies are chosen for comparison:

1. **TF-IDF + Neural Network:**
    - **Rationale:** TF-IDF provides a simple yet effective way to capture word importance across documents. It serves as a strong classical baseline, particularly for smaller datasets or domains with limited computational resources.
    - **Strengths:** Fast computation, interpretable feature importance, robust for coarse-grained sentiment classification.
    - **Limitations:** Context-independent; cannot capture polysemy, sarcasm, or nuanced sentence-level meaning.
2. **Sentence-BERT (SBERT) + Neural Network:**
    - **Rationale:** SBERT produces dense, semantically rich sentence embeddings by leveraging transformer-based contextual models. When combined with a small feedforward neural network, it allows efficient classification while capturing sentence-level nuances.
    - **Strengths:** Handles context, polysemy, and long-range dependencies; provides transferable embeddings across domains; suitable for fine-grained and cross-domain sentiment tasks.
    - **Limitations:** Slightly higher computational cost; may require careful fine-tuning for domain-specific vocabulary.

2.5.1 Proposed Approach

The proposed approach for this sentiment analysis project involves a comparative embedding strategy combined with a neural network classifier. The workflow is as follows:

1. **Embedding the Training Data:**
    - The training dataset will be processed using **two different embedding techniques**:
        - **TF-IDF (Term Frequency–Inverse Document Frequency):** Captures word importance across the corpus and provides sparse, interpretable feature vectors.
        - **Sentence-BERT (SBERT):** Generates dense, semantically rich sentence embeddings that capture contextual meaning and sentence-level nuances.
2. **Classifier Training and Tuning:**
    - For each embedding type, a **small feedforward neural network** will serve as the classification head.
    - Hyperparameters such as learning rate, batch size, number of layers, and hidden units will be systematically tuned to optimize performance for the corresponding embedding.
3. **Comparative Evaluation:**

- Both embedding-classifier combinations will be evaluated on the validation set using metrics such as accuracy, precision, recall, and F1-score.
- Misclassified examples will be analyzed to understand strengths and weaknesses of each embedding approach.

4. **Final Selection:**
    - The embedding-classifier combination that achieves the **highest overall accuracy** will be selected as the final model for sentiment prediction.
    - This ensures that the approach balances **semantic richness** (from SBERT) with **baseline interpretability and simplicity** (from TF-IDF) and leverages the embedding that best fits the dataset characteristics.

3. FEATURE REPRESENTATION AND CLASSIFIER DESIGN

The dataset, originally provided in JSON format, was first imported and loaded into the Python environment. To facilitate model training and evaluation, the original training set (7,401 samples) was further split into 70% training (5,180 samples), 15% validation (1,110 samples), and 15% test subsets (1,111 samples). This split allows the model to learn effectively from the training data, tune hyperparameters on the validation set, and evaluate final performance on unseen test data to ensure generalization.

3.1 TFIDF Vectorizer with Classification Neural Network

3.1.1 TFIDF Vectorization

TF-IDF (Term Frequency–Inverse Document Frequency) is a numerical representation of text that reflects how important a word is in a document relative to a corpus. It is built on top of the **Bag-of-Words (BoW)** model, which represents a document as an unordered collection of word counts, ignoring grammar and word order. While BoW captures word occurrence frequency, it treats all words equally, which can overemphasize common words like "the" or "and." Thus, TFIDF balances two aspects:

1. **Term Frequency (TF):** Measures how often a word appears in a document, capturing its importance within that document.
2. **Inverse Document Frequency (IDF):** Scales down the weight of words that appear frequently across many documents, reducing the influence of common but less informative words such as "the" or "is."

The TF-IDF value for a term t in a document d is computed as:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \log \frac{N}{1 + \text{DF}(t)}$$

Where N is the total number of documents, and $\text{DF}(t)$ is the number of documents containing the term t.

Advantages:

- Provides a sparse and interpretable representation of text.
- Highlights discriminative words that are important for classification.
- Simple and computationally efficient, making it suitable as a baseline feature extraction method.

Limitations:

- Context-independent: cannot distinguish between different meanings of the same word (polysemy).
- Does not capture word order or semantic relationships between words.

- Less effective for short texts or informal language where context is crucial.

3.1.2 Data Pre-Processing for TFIDF

Before converting raw text into TF-IDF vectors, the data must undergo preprocessing to clean, standardize, and structure it. Proper preprocessing ensures that the extracted features are meaningful and improves model performance.

1. Text Cleaning:

- Removing unwanted characters, symbols, or noise from the text.
- **Steps:**
    - Remove punctuation (. , ! ? : ;) and special characters (@ # $ %).
    - Convert text to lowercase to treat words like "Good" and "good" uniformly.
    - Remove numbers if they are not relevant to sentiment.
- **Python Libraries:**
    - re (regular expressions) for pattern-based cleaning.
    - string module for punctuation removal.

2. Tokenization:

- Splitting text into smaller units called tokens (usually words) for further processing.
- **Purpose:** Neural networks and TF-IDF require text in tokenized form.
- **Python Libraries:**
    - `nltk` (Natural Language Toolkit)

3. Stopword Removal:

- Stopwords are common words like "the," "is," "and" that carry little meaning.
- **Purpose:** Removing stopwords reduces noise and dimensionality, focusing on meaningful terms.
- **Python Libraries:**
    - `nltk.corpus.stopwords`

4. Lemmatization / Stemming:

- Reducing words to their base or root form (e.g., "running" → "run").
- **Purpose:** Combines variations of a word into a single feature, reducing sparsity.
- **Python Libraries:**
    - `nltk.stem.WordNetLemmatizer` for lemmatization.
    - `nltk.stem.PorterStemmer` for stemming.

3.1.3 Parameters used for TFIDF Vectorization

Embedding Size	5000	To capture a broader range of vocabulary
N-Gram Range	(1,2)	To capture both single words and short phrases
Term Frequency Scaling	Sublinear	Replacing raw term counts (tf) with 1 + log(tf).
Document Frequency (df)	Min df=2 max df=0.9	Ignores words that appear in fewer than 2 documents (too rare, usually noise) or in more than 90% of documents (too common, not discriminative).

3.1.4 TFIDF Vectorization in Python

In Python, the most widely used library for TF-IDF vectorization is **scikit-learn**, which provides efficient and easy-to-use tools for feature extraction and machine learning. The `TfidfVectorizer` class from `sklearn.feature_extraction.text` converts a collection of raw text documents into a matrix of TF-IDF features.

Key Features of TfidfVectorizer:

- Combines tokenization and TF-IDF weighting in a single step.
- Supports stopword removal, n-grams, and vocabulary size limitation.
- Produces a sparse matrix representation suitable for neural network or machine learning input.

After Embedding the Training, Validation and Test Data, the embeddings were saved as a NumPy arrays for ease of access in the future.

3.1.5 Classification Neural Network

A fully connected feedforward neural network (also known as a Multilayer Perceptron, MLP) was employed as the classification head for sentiment prediction. The model's primary objective was to learn non-linear decision boundaries over the embedding space generated using TF-IDF. By mapping these high-dimensional text embeddings to sentiment categories, the MLP effectively captured complex feature interactions that simpler models might miss. The network architecture and associated training hyperparameters were systematically tuned through experimentation to achieve a balance between classification accuracy and generalization performance.

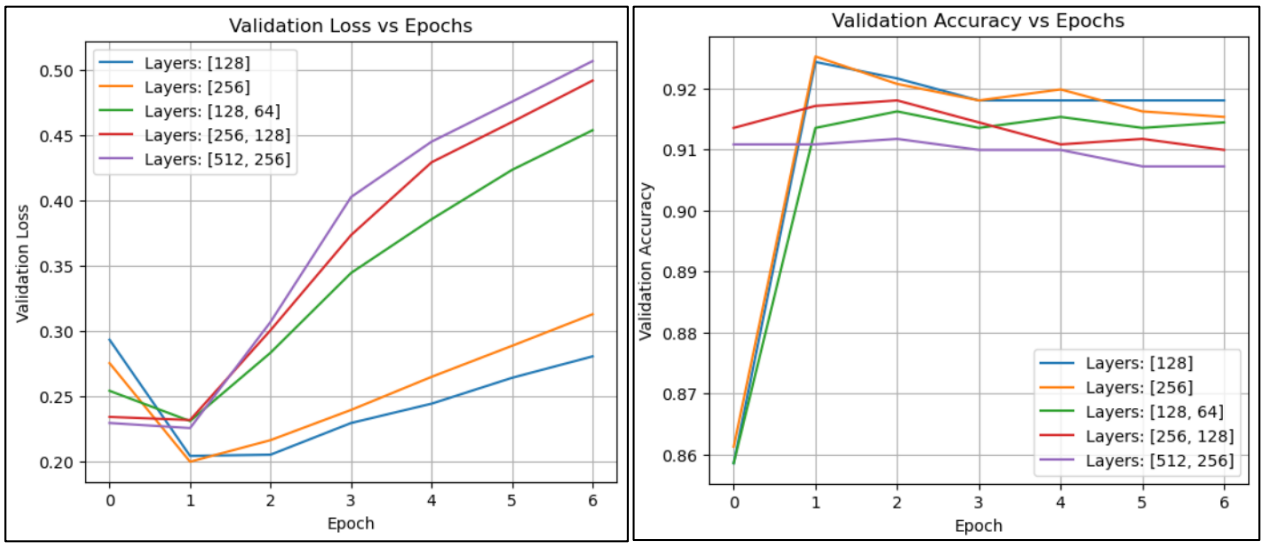
The model was trained on the training set, while performance validation was carried out on a separate validation set to guide the selection of optimal parameters. Additionally, early stopping was implemented with a patience of 5 epochs by monitoring the validation loss and restoring the best-performing model weights to prevent overfitting.

1. Tuning the Model Architecture

To determine the optimal network structure, several configurations of hidden layers and neuron counts were systematically tested. The tuning process involved experimenting with both single-layer and two-layer architectures to assess the impact of model depth and capacity on classification performance. Specifically, five configurations were evaluated: [128], [256], [128, 64], [256, 128], and [512, 256] hidden units. This progressive scaling of layer size and number allowed the model

to capture varying levels of feature abstraction while avoiding excessive complexity. The architecture yielding the highest validation accuracy with stable loss convergence was selected as the final configuration for subsequent experiments.

Plot of Validation Loss and Validation Accuracy for Tuning Model Architecture:




```
Layers [128]: Best Val Accuracy = 0.9243, Val Loss = 0.2045

Layers [256]: Best Val Accuracy = 0.9252, Val Loss = 0.2001

Layers [128, 64]: Best Val Accuracy = 0.9162, Val Loss = 0.2312

Layers [256, 128]: Best Val Accuracy = 0.9180, Val Loss = 0.2320

Layers [512, 256]: Best Val Accuracy = 0.9117, Val Loss = 0.2258
```

The validation results indicate that, for the TF-IDF feature representation, the model architecture with a **single hidden layer containing 128 neurons** achieved the best performance when other hyperparameters were held constant.

This outcome suggests that a moderately sized network provides sufficient capacity to capture the discriminative patterns present in TF-IDF embeddings without overfitting. Larger architectures, while theoretically more expressive, likely introduced unnecessary complexity, leading to slower convergence and reduced generalization due to the sparsity and high dimensionality of TF-IDF features. Hence, a simpler single-layer configuration offered the most effective balance between learning ability and robustness.

Thus, for tuning the next parameters, the model architecture was **fixed** to have a single hidden layer containing 128 neurons.

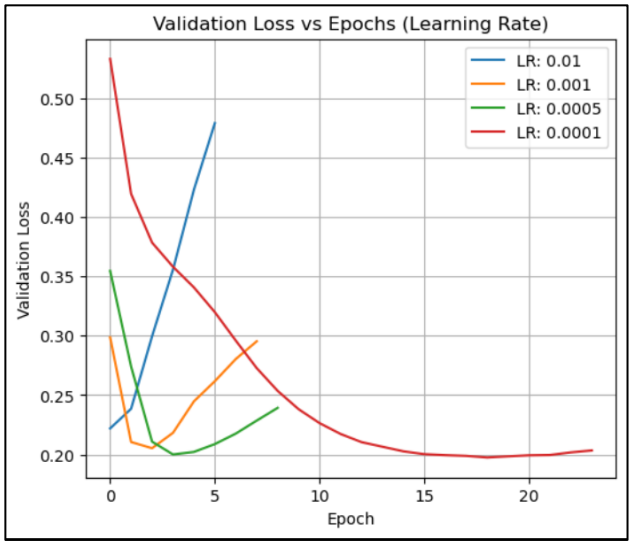
2. Tuning the Learning Rate

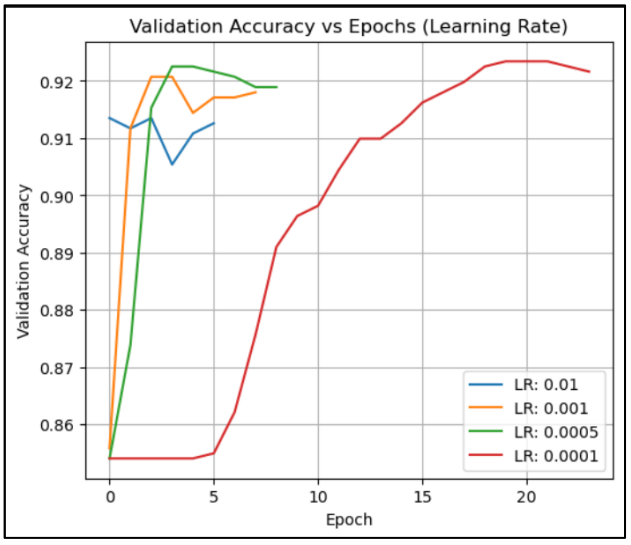
The learning rate is a key hyperparameter in neural network training that controls the step size taken by the optimizer when updating model weights during each iteration. It essentially determines how quickly or slowly a model learns from the training data.

To determine the optimal learning rate, multiple configurations were systematically tested to evaluate their effect on model convergence and stability. The tuning process involved experimenting with four learning rate values [0.1, 0.001, 0.0005, 0.0001] while keeping other parameters constant.

This range was chosen to capture both aggressive and conservative update behaviors during training**.**

Plot of Validation Loss and Validation Accuracy for Tuning Learning:






```
Learning rate 0.01: Best Val Accuracy = 0.9135, Val Loss = 0.2219

Learning rate 0.001: Best Val Accuracy = 0.9207, Val Loss = 0.2053

Learning rate 0.0005: Best Val Accuracy = 0.9225, Val Loss = 0.2000

Learning rate 0.0001: Best Val Accuracy = 0.9234, Val Loss = 0.1975
```

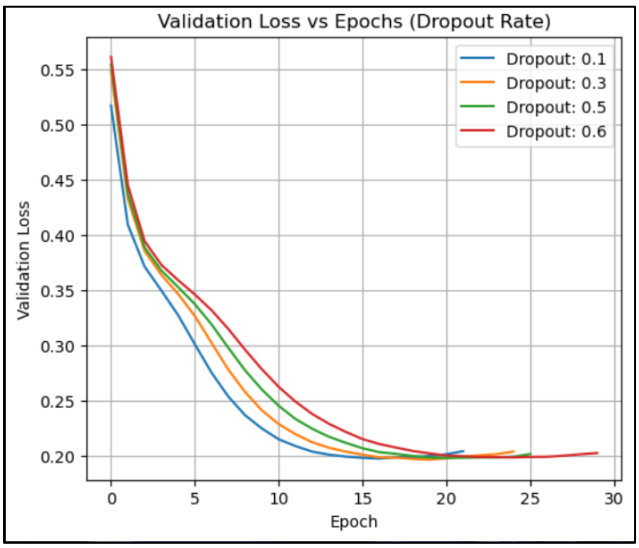
The validation results indicate that, for the TF-IDF feature representation, the model achieved the best performance with a **learning rate of 0.0001** when all other parameters were held constant. This result suggests that a smaller learning rate allowed the optimizer to make gradual and stable updates to the network's weights, preventing overshooting of the loss minima and promoting smoother convergence. In contrast, higher learning rates such as 0.1 or 0.001 led to unstable training dynamics and fluctuations in validation accuracy. The chosen value of 0.0001 thus provided the optimal trade-off between convergence stability and training efficiency. Consequently, this learning rate was fixed for subsequent hyperparameter tuning.

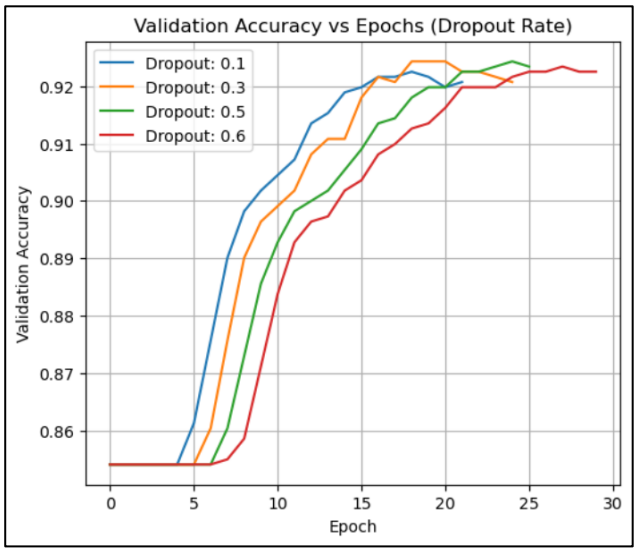
3. Dropout Rate

The dropout rate is a crucial regularization hyperparameter in neural network training that determines the fraction of neurons randomly deactivated during each training iteration. This technique prevents the network from relying too heavily on specific neurons, thereby reducing overfitting and improving generalization on unseen data.

To determine the optimal dropout rate, multiple configurations were systematically tested to evaluate their impact on model robustness and convergence. The tuning process involved experimenting with four dropout values [0.1, 0.3, 0.5, 0.6] while keeping other parameters constant. This range was selected to observe how varying degrees of regularization affect the model's learning stability and validation performance**.**

Plot of Validation Loss and Validation Accuracy for Tuning Dropout Rate:






```
Dropout 0.1: Best Val Accuracy = 0.9225, Val Loss = 0.1979
Dropout 0.3: Best Val Accuracy = 0.9243, Val Loss = 0.1970
Dropout 0.5: Best Val Accuracy = 0.9243, Val Loss = 0.1986
Dropout 0.6: Best Val Accuracy = 0.9234, Val Loss = 0.1990
```

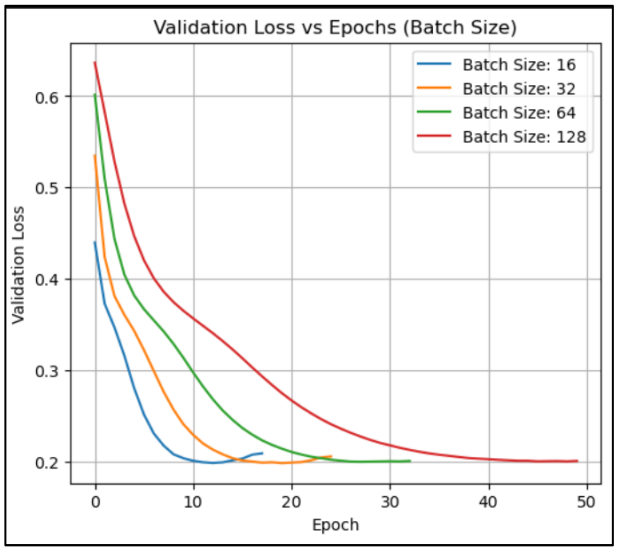
The validation results indicate that, for the TF-IDF feature representation, the model achieved the best performance with a **dropout rate of 0.3** when all other parameters were held constant. This suggests that moderately regularizing the network by randomly deactivating 30% of neurons during training was sufficient to prevent overfitting while retaining the network's capacity to learn meaningful patterns. Although other dropout values (0.1, 0.5, 0.6) produced comparable results, 0.3 provided a slightly better balance between generalization and convergence stability. Consequently, this dropout rate was fixed for subsequent hyperparameter tuning.

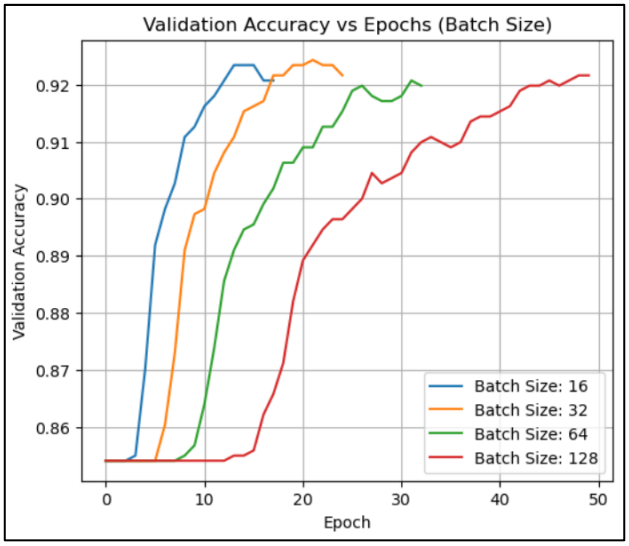
4. Batch Size

The batch size is an important training hyperparameter in neural networks that determines the number of samples processed before the model's weights are updated. Smaller batch sizes can provide more frequent updates and introduce stochasticity that may help escape local minima, but can lead to noisier gradients. Larger batch sizes produce smoother gradient estimates, improving convergence stability, but require more memory and may slow down learning dynamics.

To determine the optimal batch size, multiple configurations were systematically tested to evaluate their impact on training stability and model performance. The tuning process involved experimenting with batch sizes [16, 32, 64, 128] while keeping other parameters constant. This range was chosen to balance gradient stability, computational efficiency, and generalization capability.

Plot of Validation Loss and Validation Accuracy for Tuning Batch Size:






```
Batch Size 16: Best Val Accuracy = 0.9234, Val Loss = 0.1982

Batch Size 32: Best Val Accuracy = 0.9243, Val Loss = 0.1981

Batch Size 64: Best Val Accuracy = 0.9207, Val Loss = 0.1995

Batch Size 128: Best Val Accuracy = 0.9216, Val Loss = 0.2000
```

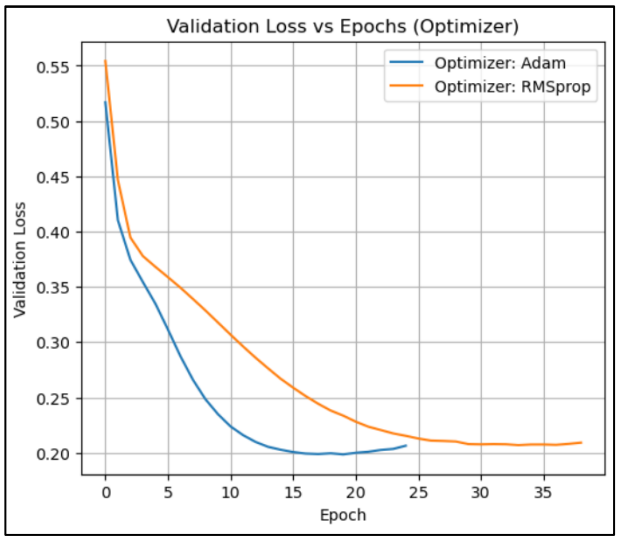
The validation results indicate that, for the TF-IDF feature representation, the model achieved the best performance with a **batch size of 32** when all other parameters were held constant. This suggests that processing 32 samples per weight update provided an optimal balance between gradient stability and training efficiency. Smaller batch sizes, while introducing more stochasticity, led to noisier gradient estimates, whereas larger batch sizes slowed convergence and required more memory without significant performance gains. Consequently, a batch size of 32 was fixed for subsequent tunings.

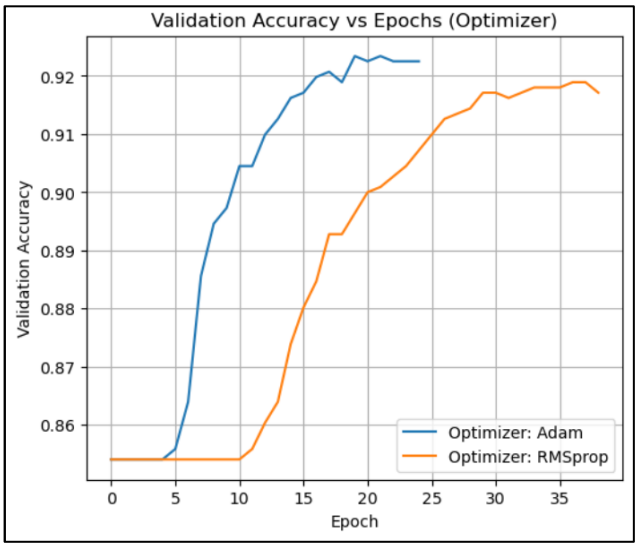
5. Optimizer

The optimizer is a key component in neural network training that determines how the model's weights are updated based on the computed gradients. Adaptive optimizers Adam and RMSprop were evaluated, as they adjust learning rates for each parameter dynamically, promoting faster and more stable convergence as compared to Stochastic Gradient Descent (SGD).

To determine the optimal optimizer, both Adam and RMSprop were systematically tested while keeping other hyperparameters constant. Their performance was assessed based on validation accuracy and loss to select the optimizer that achieves the best trade-off between convergence speed and generalization.

Plot of Validation Loss and Validation Accuracy for Tuning Optimizer:






```
Optimizer Adam: Best Val Accuracy = 0.9234, Val Loss = 0.1985

Optimizer RMSprop: Best Val Accuracy = 0.9189, Val Loss = 0.2069
```

The validation results indicate that **Adam Optimizer** performs the best due to its adaptive learning rate mechanism, which adjusts the step size for each parameter individually, combining the benefits of momentum and RMSProp. Adam consistently produced better validation accuracy and smoother training curves compared to RMS Prop. Therefore, Adam was fixed as the optimizer.

6. Final Hyperparameters for the Classification Model using TF-IDF Embeddings

Hyper Parameter	Optimal/ Chosen Value
Network Architecture	Single Hidden Layer with 128 Neurons
Learning Rate	0.0001
Dropout Rate	0.3
Batch Size	32
Optimizer	Adam

7. Performance Metrics of the Model on Testing Dataset

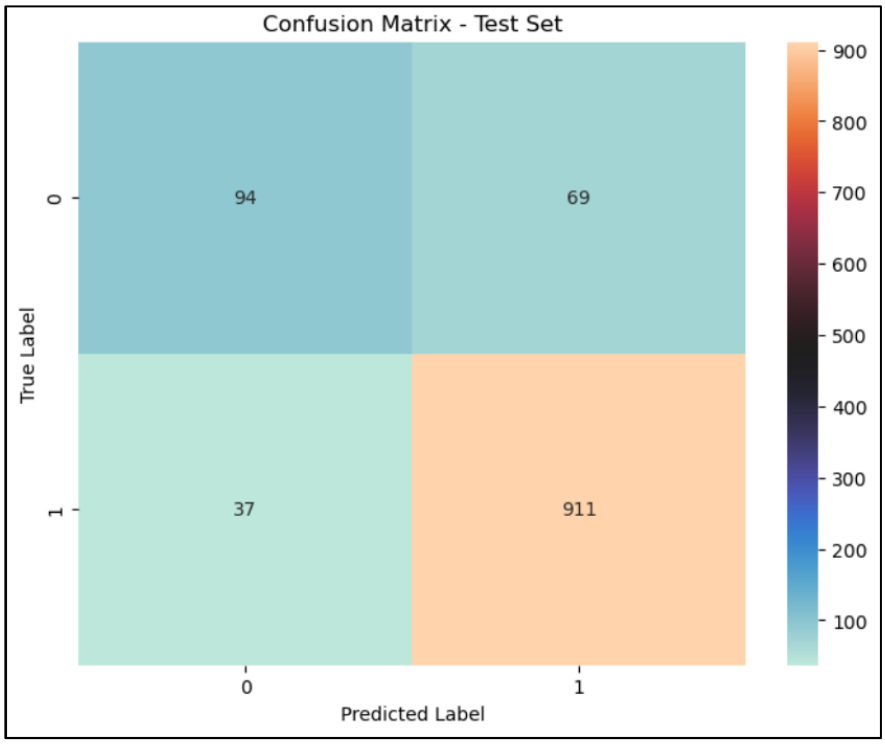
The final neural network classifier was designed using the optimal hyperparameter values determined through systematic tuning: a single hidden layer with 128 neurons, learning rate of 0.0001, dropout rate of 0.3, batch size of 32, and Adam optimizer. Using this configuration, the model was trained on the combined training and validation embeddings to fully leverage the labeled data for learning. After training, the model's performance was evaluated on the held-out test embeddings to obtain unbiased estimates of its classification accuracy, loss, and other relevant metrics. This evaluation provides a realistic assessment of how the model generalizes to unseen data.

Performance Metrics:

Accuracy	90.46 %
F1 Score	0.9002

Test Dataset Distribution:
Class 0: 163
Class 1: 948

Confusion Matrix:



The fully tuned TF-IDF + MLP model achieved a test accuracy of 90.46% and an F1 score of 0.9002, indicating strong overall performance in sentiment classification.

- The model is highly effective at identifying positive sentiment (class 1), with only a small number of false negatives (37).
- The performance is slightly weaker on negative sentiment (class 0), with 69 false positives, which may be due to class imbalance in the dataset or the sparse nature of TF-IDF features.
- Overall, the classifier demonstrates robust generalization on unseen test data, with a balanced trade-off between precision and recall, as reflected in the high F1 score.

3.2 Sentence BERT (SBERT) Embeddings with Classification Neural Network

3.2.1 SBERT Embedding

Sentence-BERT (SBERT) is a modification of the BERT (Bidirectional Encoder Representations from Transformers) model designed to efficiently generate sentence-level embeddings suitable for tasks like semantic similarity and classification. While BERT provides powerful contextual word embeddings, using it directly for sentence-level tasks requires pairwise comparisons between embeddings, which is computationally expensive. SBERT addresses this by introducing a Siamese network architecture, allowing sentences to be encoded into fixed-size dense vectors that can be compared using cosine similarity or fed directly into downstream classifiers.

SBERT embeddings capture both semantic meaning and contextual relationships, overcoming the limitations of traditional sparse representations like TF-IDF that ignore word order and semantics. These dense embeddings are particularly effective for tasks where context and meaning of the entire sentence or document are crucial, such as sentiment classification.

Advantages:

- Captures semantic and syntactic information from entire sentences.
- Produces fixed-size embeddings, independent of sentence length.
- Efficient for real-time or large-scale comparisons due to pre-computed embeddings.
- Works well with deep learning classifiers such as MLPs.

Limitations:

- Computationally heavier than TF-IDF for embedding generation.
- May require GPU acceleration for faster processing.
- Dense embeddings are less interpretable compared to sparse TF-IDF vectors.

3.2.2 Data Pre-Processing for SBERT

Before feeding sentences to SBERT, minimal preprocessing is required since the model is robust to case, punctuation, and common language noise. However, some cleaning improves efficiency and quality:

1. **Text Cleaning:**
    - Remove excessive whitespace and non-informative symbols (optional).
    - Keep sentences intact to preserve context.
    - Python Libraries: re for regex-based cleaning.
2. **Tokenization:**
    - Handled internally by SBERT's pre-trained tokenizer.
    - Converts sentences into subword tokens compatible with the model.

3. **Lowercasing / Casing:**
    - Depending on the pre-trained SBERT model, casing may be preserved or normalized.
    - Python Libraries: transformers library from HuggingFace.
4. **Stopword Removal / Lemmatization:**
    - Generally not required, as SBERT embeddings are context-aware and handle stopwords effectively.

3.2.3 Parameters Used for SBERT Embedding

Pre-trained Model	all-MiniLM-L6-v2
Embedding Size	384 (for MiniLM)
Pooling Strategy	Mean pooling of token embeddings
Max Sentence Length	128 tokens

3.2.4 SBERT Embedding in Python

SBERT embeddings can be generated using the sentence-transformers library in Python, which provides an easy interface to encode text into fixed-size vectors.

Key Features of SBERT:

- Combines contextual embeddings with mean/max pooling for sentence representation.
- Supports batch processing for efficiency.
- Produces dense, semantic-aware embeddings suitable for MLP or other classifiers.

After embedding the training, validation, and test data, the vectors were saved as NumPy arrays for easy reuse in downstream classification experiments.

3.2.5 Classification Neural Network

For the SBERT embeddings, the same fully connected feedforward neural network (MLP) architecture used with TF-IDF was employed as the classification head. The network was systematically tuned for the following hyperparameters, in the same manner as for TF-IDF embeddings:

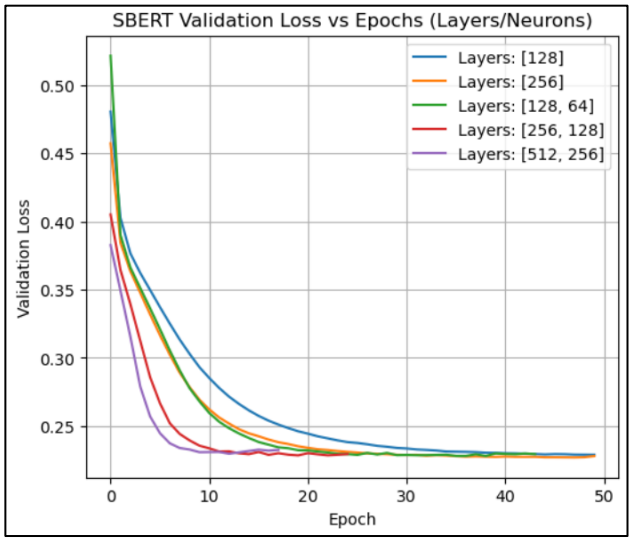
- Model Architecture: Number of hidden layers and neurons per layer
- Learning Rate: Step size for optimizer updates
- Dropout Rate: Fraction of neurons randomly deactivated to prevent overfitting
- Batch Size: Number of samples processed per weight update
- Optimizer Choice: Adam optimizer for stable and efficient convergence

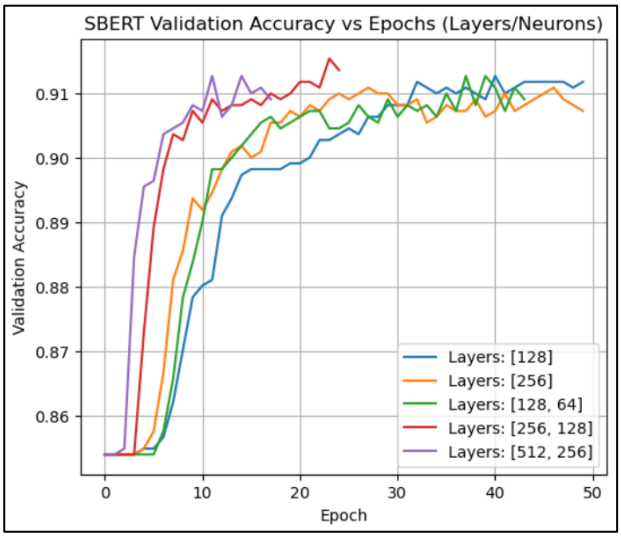
Each hyperparameter was optimized based on validation performance, ensuring that the SBERT embeddings were leveraged effectively to maximize classification accuracy and generalization. The results of this tuning process for SBERT embeddings are summarized below.

1. Model Architecture

Architecture Tested: [128], [256], [128, 64], [256, 128], and [512, 256]

Plot of Validation Loss and Validation Accuracy for Network Architecture:





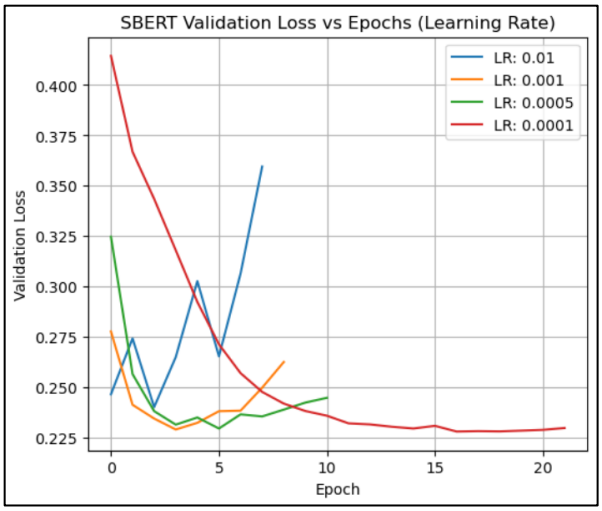

```
Layers [128]: Best Val Accuracy = 0.9126, Val Loss = 0.2290

Layers [256]: Best Val Accuracy = 0.9108, Val Loss = 0.2271

Layers [128, 64]: Best Val Accuracy = 0.9126, Val Loss = 0.2281

Layers [256, 128]: Best Val Accuracy = 0.9153, Val Loss = 0.2286

Layers [512, 256]: Best Val Accuracy = 0.9126, Val Loss = 0.2298
```

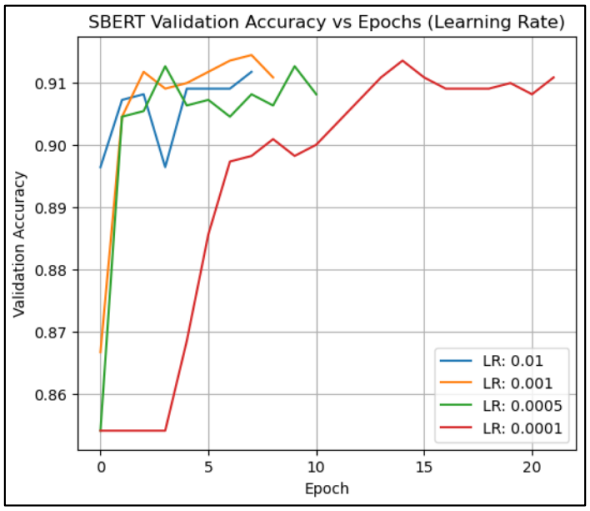
For the SBERT embeddings, the validation results indicated that a **two-layer architecture with hidden layer sizes [512, 256]** achieved the best performance. This configuration likely provided the optimal balance between model capacity and generalization: the first layer with 512 neurons captured complex patterns and interactions within the dense SBERT embeddings, while the second layer with 256 neurons refined these representations to produce more discriminative features for classification. Compared to a single-layer or smaller network, this deeper and wider architecture was better suited to leverage the rich semantic information encoded in SBERT, allowing the network to model subtle nuances in sentiment without overfitting.

2. Learning Rate

Learning Rates Tested: [0.01, 0.001, 0.0005, 0.0001]

Plot of Validation Loss and Validation Accuracy for Learning Rates:





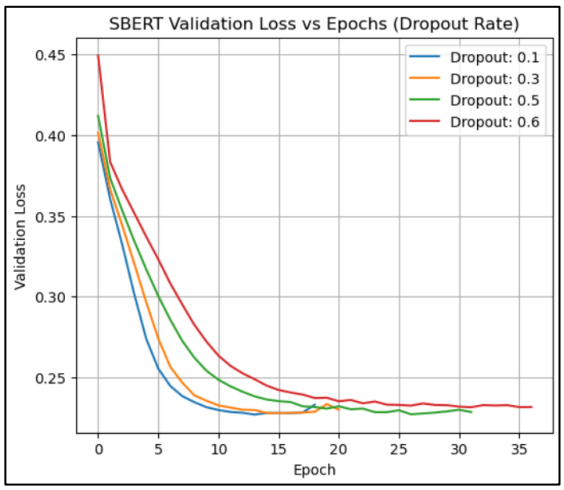

```
Learning rate 0.01: Best Val Accuracy = 0.9117, Val Loss = 0.2402

Learning rate 0.001: Best Val Accuracy = 0.9144, Val Loss = 0.2291

Learning rate 0.0005: Best Val Accuracy = 0.9126, Val Loss = 0.2296

Learning rate 0.0001: Best Val Accuracy = 0.9135, Val Loss = 0.2281
```

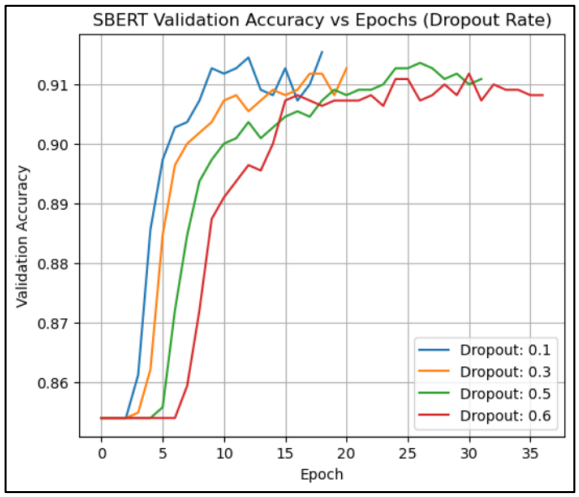
For the SBERT embeddings, the validation results indicated that a **learning rate of 0.0001** yielded the best performance when all other parameters were held constant. This small learning rate allowed the optimizer to make gradual and stable updates to the network's weights, preventing overshooting of the loss minima and promoting smooth convergence. Larger learning rates led to unstable training dynamics, causing fluctuations in validation accuracy and slower convergence. The chosen value of 0.0001 thus provided the optimal trade-off between convergence stability and training efficiency for the SBERT-based classifier

3. Dropout Rate

Dropout Rates Tested: [0.1, 0.3, 0.5, 0.6]

Plot of Validation Loss and Validation Accuracy for Dropout Rates:






```
Dropout 0.1: Best Val Accuracy = 0.9153, Val Loss = 0.2269
```

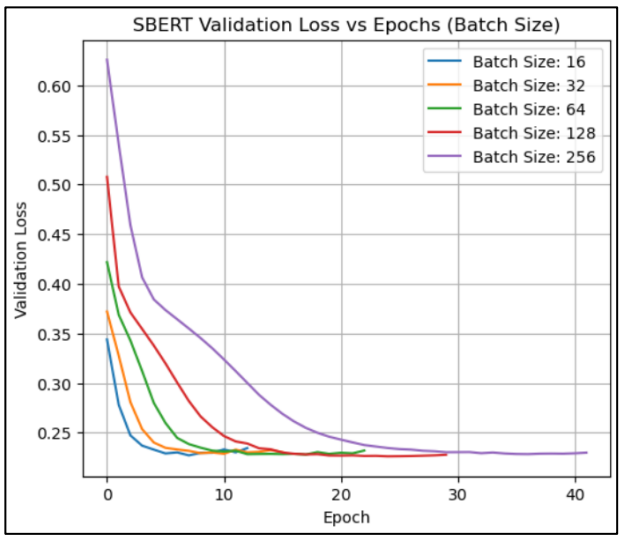
```
Dropout 0.3: Best Val Accuracy = 0.9126, Val Loss = 0.2278

Dropout 0.5: Best Val Accuracy = 0.9135, Val Loss = 0.2271

Dropout 0.6: Best Val Accuracy = 0.9117, Val Loss = 0.2314
```

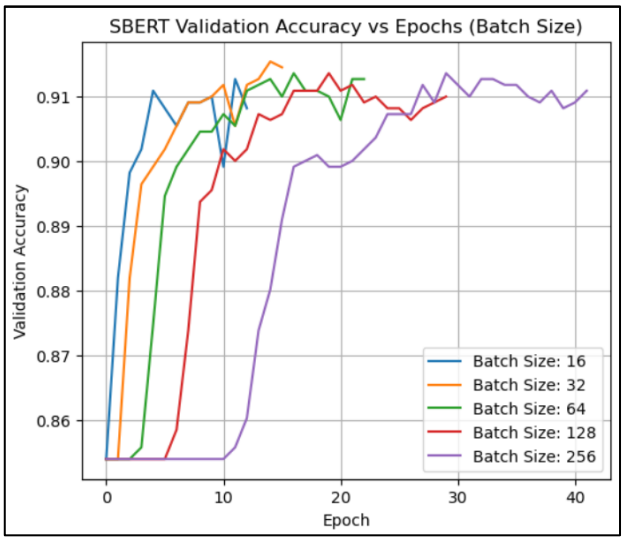
For the SBERT embeddings, the model achieved the best performance with a **dropout rate of 0.1** when all other hyperparameters were held constant. A relatively low dropout rate was sufficient because SBERT embeddings are dense, semantically rich, and already capture meaningful relationships between words, reducing the risk of overfitting. Higher dropout values, such as 0.3 or 0.5, tended to overly regularize the network, slightly limiting its ability to fully leverage the rich information in the embeddings. Consequently, a dropout rate of 0.1 provided an optimal balance between preventing overfitting and preserving the network's capacity to learn from SBERT features.

4. Batch Size

Batch Sizes Tested: [16, 32, 64, 128, 256]

Plot of Validation Loss and Validation Accuracy for Batch Sizes:





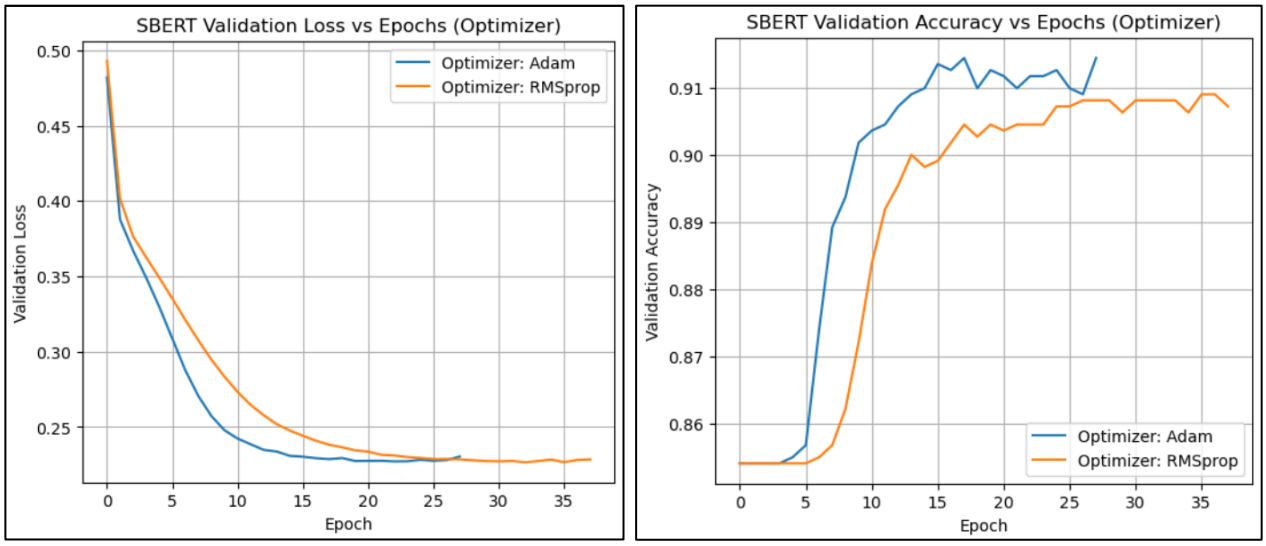

```
Batch Size 16: Best Val Accuracy = 0.9126, Val Loss = 0.2271

Batch Size 32: Best Val Accuracy = 0.9153, Val Loss = 0.2287

Batch Size 64: Best Val Accuracy = 0.9135, Val Loss = 0.2279

Batch Size 128: Best Val Accuracy = 0.9135, Val Loss = 0.2262

Batch Size 256: Best Val Accuracy = 0.9135, Val Loss = 0.2284
```

For the SBERT embeddings, the model achieved the best performance with a **batch size of 32** while keeping all other hyperparameters constant. This smaller batch size provided a good balance between gradient stability and stochasticity, allowing the optimizer to make more frequent updates that improved convergence on the dense, semantically rich SBERT embeddings. Larger batch sizes tended to produce smoother but less responsive updates, potentially slowing adaptation to subtle patterns in the data. Therefore, a batch size of 32 offered the optimal trade-off between training efficiency, convergence stability, and effective utilization of the embedding information.

5. Optimizer

Optimizers Tested: RMSprop, Adam

Plot of Validation Loss and Validation Accuracy for Optimizers:




```
Optimizer Adam: Best Val Accuracy = 0.9144, Val Loss = 0.2271

Optimizer RMSprop: Best Val Accuracy = 0.9090, Val Loss = 0.2266
```

The validation results indicate that, for the SBERT feature representation, the model achieved the best performance with the **Adam optimizer** when all other parameters were held constant. This suggests that Adam's combination of momentum and adaptive learning rates allowed the network to converge efficiently while effectively handling the dense and contextual SBERT embeddings. Although RMSprop produced a slightly lower validation loss, Adam's higher validation accuracy demonstrates its superior ability to generalize on the sentiment classification task. Consequently, Adam was selected as the optimizer for subsequent training and evaluation experiments.

6. Final Hyperparameters for the Classification Model using SBERT Embeddings

Hyper Parameter	Optimal/ Chosen Value
Network Architecture	Two Hidden Layers with [256,128] Neurons
Learning Rate	0.0001
Dropout Rate	0.1
Batch Size	32
Optimizer	Adam

7. Performance Metrics of the Model on Testing Dataset

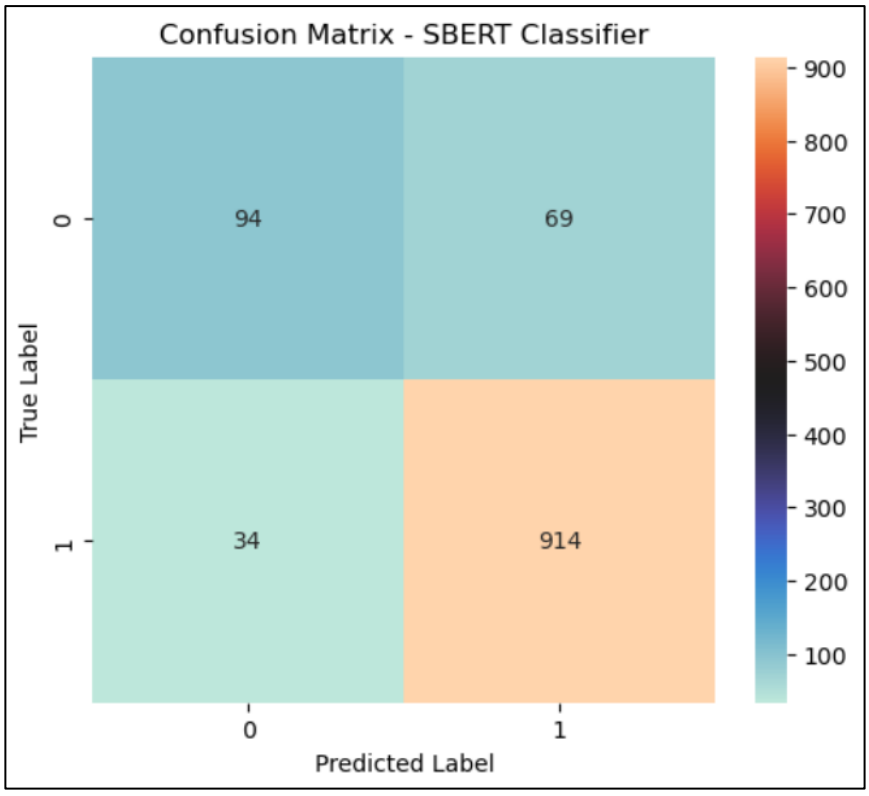
The final neural network classifier was designed using the optimal hyperparameter values determined through systematic tuning: two hidden layers with [512 , 256] neurons, learning rate of 0.0001, dropout rate of 0.3, batch size of 32, and Adam optimizer. Using this configuration, the model was trained on the combined training and validation embeddings to fully leverage the labeled data for learning. After training, the model's performance was evaluated on the held-out test embeddings to obtain unbiased estimates of its classification accuracy, loss, and other relevant metrics. This evaluation provides a realistic assessment of how the model generalizes to unseen data.

Performance Metrics:

Accuracy	90.73 %
F1 Score	0.9026

Test Dataset Distribution:
Class 0: 163
Class 1: 948

Confusion Matrix:



The results indicate that the fully tuned SBERT-based classifier performed very well on the test set, achieving a high accuracy of 90.73% and an F1 score of 90.26%, demonstrating a strong balance between precision and recall. The confusion matrix shows that the model correctly classified the majority of positive and negative samples, with only a small number of misclassifications (69 false positives and 34 false negatives). This suggests that the network effectively leveraged the rich

semantic information in SBERT embeddings to capture nuanced patterns in the text, providing robust sentiment predictions with minimal bias toward either class. Overall, the model demonstrates both high predictive performance and strong generalization on unseen data.

3.3 Conclusion and Model Selection

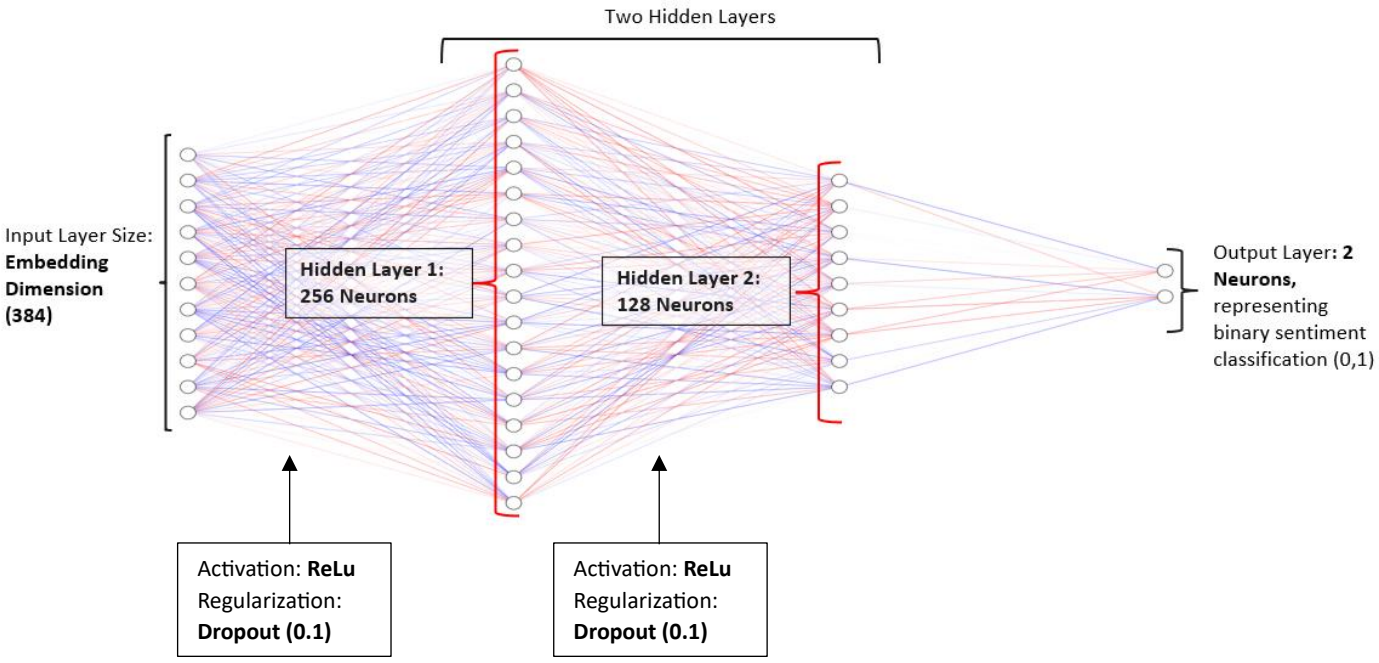
The TF-IDF-based classifier demonstrated robust performance, achieving a test accuracy of 90.46% and an F1 score of 0.9002. When compared to the SBERT-based model, which achieved a slightly higher accuracy of 90.73% and an F1 score of 0.9026, it is evident that SBERT only offered a modest improvement. This limited margin can be explained by several factors:

1. **Dataset Size and Complexity:** The dataset consisted of ~7,400 reviews labeled as positive or negative. While sufficient for learning general patterns, this size may not fully exploit SBERT's ability to capture nuanced semantic relationships, which become more advantageous on larger or more complex datasets.
2. **Nature of the Task**: Sentiment classification with binary labels is relatively straightforward. TF-IDF captures the presence and importance of key sentiment words effectively, so the richer semantic embeddings of SBERT provide only incremental gains.
3. **High-Dimensional Feature Representations:** Although SBERT embeddings encode context and word interactions better than TF-IDF, the dense representations may contain redundant information for a binary sentiment task. TF-IDF, being sparse, already highlights the most discriminative terms, allowing the classifier to perform strongly.
4. **Model Capacity and Training:** Both classifiers were trained with optimally tuned neural networks. Given the modest complexity of the task, the neural network was sufficient to extract meaningful patterns from both types of embeddings, reducing the advantage of SBERT's contextual information.

In summary, SBERT performed slightly better, reflecting its ability to encode richer semantic features, but the gap is small because the task and dataset allow TF-IDF to capture most of the discriminative information needed for effective sentiment classification.

Thus, we choose SBERT and its classifier Neural Network for the Classification of the Test dataset, however, TF-IDF with its classifier will offer comparative results while being computationally efficient.

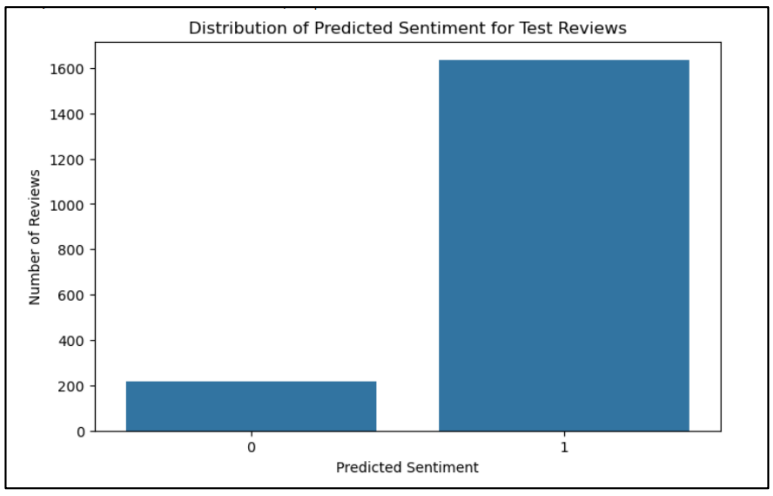
The final Architecture of the Neural Network is shown by the diagram below:



Chosen Hyperparameters/ Configuration for the Neural Network:

Hyperparameters/ Configuration	Optimal/ Chosen Value
Network Architecture	Two Hidden Layers with [256,128] Neurons
Learning Rate	0.0001
Dropout Rate	0.1
Batch Size	32
Optimizer	Adam
Activation within the hidden layers	ReLu
Activation function at the output layer	SoftMax
Epoch	50
Call backs	Early Stopping with a patience of 5, monitoring validation loss

Finally, the test data set was embedded and passed through the classification neural network to obtain the prediction results.



4. STRENGTHS AND WEAKNESSES OF THE MODEL ON THE TEST DATASET

Since the test set was unlabeled, a manual evaluation process was carried out to assess the correctness of the model's predictions. The predicted sentiments were first paired with their corresponding review texts and then analyzed using large language models (ChatGPT and Gemini) to identify potential misclassifications or ambiguous cases. The flagged samples were subsequently reviewed and verified manually. This combined automated–human validation approach allowed for a deeper understanding of where the model performs well and where it struggles, revealing key strengths and weaknesses in handling different types of reviews.

4.1 Examples of the correctly predicted review sentiments:

1. **Review:** *Finally I have found a quality brand of swimsuit I can order on line and know it will fit. I'm a daily swimmer who is long-bodied. This particular style fit great when it arrived; it arrived quickly within four days; shipping was free; and the lycra material is the most resilient I know given the chlorine bashing my suits get. Please continue to sell this particular item at a price lower than the sports stores. Thanks!!*
   Predicted Sentiment: **Positive**
   True Sentiment: Positive
   Inference: The model correctly identified this review as positive. The strong presence of explicit positive phrases such as "quality brand," "fit great," "arrived quickly," and "resilient material" provided clear sentiment cues. The overall tone conveys satisfaction and gratitude, both of which are well captured by the SBERT embeddings' contextual understanding.This demonstrates the model's strength in handling clearly positive sentiment expressed through direct language

2. **Review:** *I was thinking about getting one of those fancy motorized tie racks with a gazillion features...and then read all the bad reviews...and decided to stick to the bare basic design. I bought two, all the ties hang nice&neat in the closet, and if we move or change closets, I don't have to worry about anything (especially the batteries dying)*
   Predicted Sentiment: **Positive**
   True Sentiment: Positive
   Inference: Although the review begins with some neutral and slightly skeptical phrasing, the overall sentiment turns clearly positive toward the end. Phrases like "all the ties hang nice & neat" and "don't have to worry about anything" express satisfaction with the product's simplicity and reliability. This highlights the model's contextual understanding of sentiment progression within a review. Despite mixed phrasing at the start, the SBERT-based classifier correctly weighted the overall tone toward satisfaction. This shows that the model does not rely solely on individual words but captures sentence-level meaning and overall narrative sentiment effectively.

3. **Review:** *Does not say on amazons site that these are shipped in narrow widths only. Supplier does not carry the wide ones. Birkenstocks are normally wide unless you specify narrow widths. Not here. No exchange possible. Now I'm stuck with shoes that don't fit and midwest sports couldn't care less. I won't ever order from them again*
   Predicted Sentiment: **Negative**
   True Sentiment: Negative
   Inference: The model accurately detected negative sentiment despite the presence of mixed factual and emotional content. Phrases like "stuck with shoes that don't fit," "couldn't care less," and "won't ever order again" strongly indicate frustration and dissatisfaction.This shows the model's strength in understanding emotionally charged and complaint-heavy text, even when combined with neutral descriptive phrases. It highlights the model's ability to capture tone intensity and contextual polarity effectively, rather than relying solely on surface-level word counts.

4. **Review:** *The sole is high-profile and well-cushioned. However, the top material is a soft and flimsy towel-like material that does not provide any lateral support for the foot. With any walking pressure to the side... the foot is allowed to slide and 'fall off' of the sole in a sprain-your-ankle sort of way. I felt very at-risk when I walked in them.*
   Predicted Sentiment: **Negative**
   True Sentiment: Negative
   Inference: Even though the review begins with a mildly positive statement ("well-cushioned"), the model correctly identified the overall negative sentiment. Words such as "flimsy," "does not provide support," "fall off," and "at-risk" clearly dominate the sentiment, indicating disappointment and concern.This demonstrates the model's ability to handle mixed sentiment reviews where both positive and negative descriptors appear.

4.2 Examples of the incorrectly predicted review sentiments:

1. **Review:** *The socks are very lightweight, probably would be ok for summer use but not for work or books. They were just ok. I would not purchase again.*
   Predicted Sentiment: Positive
   True Sentiment: Negative
   Inference: The model was likely influenced by neutral or mildly positive phrases such as "lightweight" and "ok," while failing to capture the decisive negative intent expressed in "I would not purchase again." This shows that the model sometimes struggles when sentiment cues appear near the end of the review or when overall tone is subtly dissatisfied rather than strongly emotional.

2. **Review:** *Save your money and lock your luggage with cheap lock ties. Our TSA lock was cut the very first time we used it. Turns out the ENTIRE terminal has only 1 set of key so screener just don't bother /have the time to unlock this approved lock*
   Predicted Sentiment: Positive
   True Sentiment: Negative
   Inference: Despite the strong negative language ("Save your money," "cut," "don't bother"), the model classified this as positive, likely due to sentence complexity and lack of overtly

emotional words. This indicates that the classifier may underperform on complaint-style reviews that express frustration indirectly through sarcasm or narrative detail instead of explicit negative adjectives.

3. **Review:** *If you have block feet like I do you know how hard it is to find wide shoes that are not awful looking. This shoe is good for casual wear, handsome but not an athletic shoe... The leather took a little while to break-in but once it did these became one of my favorite pairs of shoes*
   Predicted Sentiment: Negative
   True Sentiment: Positive
   Inference: This review expresses overall satisfaction, but the model may have been misled by a mixed structure, starting with complaints about difficulty finding good shoes and mentioning a "break-in" period. It demonstrates that the classifier sometimes struggles to interpret contrastive sentiment flow, where early negative context transitions into a positive conclusion.

4. **Review:** *They're cool, just not as comfortable as I thought they would be. I don't like that thingy holding my toe, unlike other sandals that don't have a string in between.*
   Predicted Sentiment: Positive
   True Sentiment: Negative
   Inference: The review clearly reflects mild disappointment, but the model likely overemphasized the phrase "They're cool," which is a positive cue at the start. This suggests a positional bias, where early positive expressions outweigh later negative opinions. It also highlights that the model can miss low-intensity dissatisfaction when sentiment is mixed and not strongly polarized.

5. EFFECT OF FEATURE FORMAT ON ACCURACY AND COMPUTATIONAL EFFICIENCY

In natural language processing (NLP), feature representation refers to the process of converting raw text into numerical vectors that can be processed by machine learning algorithms. Since models cannot directly interpret words or sentences, text must be transformed into a structured, mathematical form that preserves meaningful information about the content, context, and relationships between words.

The choice of feature representation significantly impacts both the performance and efficiency of a model. Simpler representations, such as Bag-of-Words or TF-IDF, are computationally inexpensive but often fail to capture the semantic and contextual nuances of text. On the other hand, advanced representations, like contextual embeddings from transformer-based models, provide richer semantic understanding and generally lead to higher accuracy, but they require considerably more computational resources, memory, and inference time. Therefore, there is an inherent trade-off between the expressiveness of the features and the associated resource consumption, and selecting an appropriate representation depends on the complexity of the task, the size of the dataset, and available computational capabilities.

5.1 Basic Feature Formats

5.1.1 Bag-of-Words (BoW)

Description: The Bag-of-Words model represents each document as a vector of word counts. It completely ignores word order and context, treating the document as an unordered collection of words. Each dimension of the vector corresponds to a unique word in the vocabulary, and the value indicates the frequency of that word in the document.

Resource Consumption:

- Computationally inexpensive and easy to implement.
- For large vocabularies, the resulting vectors are sparse and high-dimensional, which can increase memory usage and storage requirements.

Accuracy Implications:

- Captures basic word occurrence information, which is sufficient for simple classification tasks.
- Cannot capture semantic meaning or contextual relationships between words; polysemous words (words with multiple meanings) are not differentiated.

5.1.2 TF-IDF (Term Frequency–Inverse Document Frequency)

Description: TF-IDF improves upon BoW by weighting each word's frequency according to how common or rare it is across the corpus. Words that are frequent in a particular document but rare across the dataset are given higher importance, making them more discriminative for classification.

Resource Consumption:

- Slightly higher than BoW due to the additional computation of inverse document frequency (IDF).

- Still manageable on typical CPUs, and the vectors remain sparse.

Accuracy Implications:

- Generally outperforms BoW for classification tasks by emphasizing informative words.
- Remains context-independent and cannot capture word order or deeper semantic meaning.

5.2 Intermediate Feature Formats

5.2.1 Word Embeddings (Word2Vec, GloVe)

Description: Word embeddings are dense vector representations that map each word to a continuous vector in a high-dimensional space. Models like Word2Vec and GloVe learn embeddings such that semantically similar words are close together in this space. Unlike sparse BoW or TF-IDF vectors, embeddings capture underlying semantic relationships between words, allowing the model to understand similarity and analogies (e.g., "king" – "man" + "woman" ≈ "queen").

Resource Consumption:

- **Memory:** Requires storing a dense embedding matrix of size (vocabulary × embedding_dim). For large vocabularies, this can become significant, but still smaller than full one-hot or TF-IDF matrices.
- **Computation:** If pretrained embeddings are used, memory is the primary cost. Training embeddings from scratch is computationally expensive, requiring large corpora and multiple epochs for meaningful representations.
- **Inference:** Using embeddings is efficient once loaded; vector lookup is fast.

Accuracy Implications:

- Captures semantic relationships between words, improving model understanding compared to BoW or TF-IDF.
- Enables better generalization on unseen words with similar meanings.
- Limitation: Aggregating word embeddings to represent entire sentences or documents often ignores word order and long-range dependencies, which can reduce accuracy in tasks sensitive to context.

5.2.2 Sentence Embeddings (InferSent, SBERT)

Description: Sentence embeddings represent entire sentences, paragraphs, or documents as dense vectors. Models like InferSent and SBERT generate embeddings that capture both semantic meaning and contextual information, including word order and syntactic structure. These embeddings are particularly effective for tasks such as sentiment classification, semantic similarity, and entailment.

Resource Consumption:

- **Memory:** Significantly higher than word embeddings because the models often contain millions of parameters to encode context and meaning.

- **Computation:** Generating embeddings is more computationally intensive. Pretrained models help reduce training cost, but inference can still be slower compared to word embeddings or TF-IDF, especially for large datasets.
- **Scalability:** Deploying sentence embeddings for very large datasets or real-time applications requires consideration of GPU memory and batch processing.

Accuracy Implications:

- Provides superior performance in classification, semantic understanding, and other NLP tasks because it captures context, polysemy, and nuanced meaning.
- Handles sentences with multiple meanings more effectively than word-level embeddings.
- Particularly beneficial for datasets where sentence structure and subtle context influence the label, such as sentiment classification or natural language inference.
- Trade-off: The increased accuracy comes at the cost of greater computational resources, both in memory and processing time.

This highlights the progression from word-level to sentence-level embeddings, as the feature representation becomes richer and more context-aware, the model's accuracy improves, but the computational burden also increases substantially.

5.3 Advanced / Contextual Feature Formats

5.3.1 Transformer-based Embeddings (BERT, RoBERTa, GPT)

Description: Transformer-based embeddings leverage attention mechanisms to generate deep contextual representations of text. Unlike static embeddings, these models produce different embeddings for the same word depending on its context within a sentence. Models like BERT, RoBERTa, and GPT encode not only word meaning but also syntactic structure, long-range dependencies, and semantic nuances, making them highly effective for tasks such as sentiment analysis, question answering, and textual entailment.

Resource Consumption:

- **Memory:** Very high. Transformer models have millions to billions of parameters, requiring substantial RAM or GPU memory, especially for longer sequences.
- **Computation:** Inference is computationally intensive due to self-attention layers and multiple transformer blocks. Generating embeddings for large datasets can be slow without batching or GPU acceleration.
- **Scalability:** Deploying transformers at scale demands optimized hardware (GPUs/TPUs), mixed-precision computation, and careful batching strategies to handle memory constraints.

Accuracy Implications:

- Consistently provides state-of-the-art performance on semantic and sentiment tasks.
- Capable of capturing subtle nuances, polysemy, and context-dependent meaning, outperforming both static word embeddings and sentence embeddings on complex tasks.
- Generalizes well to unseen or out-of-domain text due to pretraining on large corpora.

- Trade-off: The exceptional accuracy comes at a significant cost in resource consumption and inference latency, which may be prohibitive for real-time or resource-limited applications.

5.3.2 Fine-tuning Transformers

Description: Fine-tuning involves adapting a pretrained transformer to a specific downstream classification task by training its parameters on task-specific data. This allows the model to specialize its representations for the nuances of the target dataset, such as sentiment in product reviews or hotel feedback.

Resource Consumption:

- **Training:** Extremely high computational cost. Fine-tuning requires GPUs and can take hours to days depending on dataset size and model complexity.
- **Memory:** High memory usage due to storing gradients for backpropagation across all transformer layers.
- **Inference:** Slightly more efficient than training but still heavier than static embeddings, especially for long sequences.

Accuracy Implications:

- Can achieve state-of-the-art results, outperforming traditional embeddings or shallow classifiers.
- Captures task-specific nuances, improving classification accuracy significantly on challenging datasets.
- Risk of overfitting on small datasets; careful regularization or parameter freezing may be needed.
- Often overkill for simple datasets, where lighter embeddings or classical methods may suffice while being more efficient.

Advanced transformer-based embeddings and fine-tuning provide the most powerful and context-aware representations in NLP, offering superior accuracy and generalization. However, these gains come with dramatically higher memory, compute, and deployment costs, necessitating careful consideration of the project's resources and dataset complexity.

Feature Type	Description	Accuracy / Effectiveness	Resource Consumption	Practical Considerations
Bag-of-Words (BoW)	Counts word frequency, ignores order/context	Basic; works for simple classification	Very low; sparse matrices, CPU-friendly	Good for small datasets, low-resource settings; misses semantics
TF-IDF	Weighted BoW emphasizing rare/discriminative words	Better than BoW; context-independent	Low; slightly higher than BoW	Suitable for straightforward classification; still limited semantic understanding
Word Embeddings (Word2Vec, GloVe)	Dense vectors capturing semantic similarity	Moderate; captures similarity between words	Moderate memory; can train or use pretrained	Suitable when semantics matter; faster with pretrained embeddings
Sentence Embeddings (InferSent, SBERT)	Embeddings at sentence/document level capturing context	High; captures context and polysemy	Higher memory/computation; pretrained helps	Best for sentiment/semantic tasks with moderate compute resources
Transformer-based Embeddings (BERT, RoBERTa, GPT)	Deep contextual embeddings; full sentence context	Very high; state-of-the-art	Very high GPU/CPU usage, slow inference	Ideal for large/complex datasets; requires substantial compute
Fine-tuned Transformers	Adapting pretrained transformers to specific task	Maximum; can achieve SOTA	Extremely high; requires GPUs/TPUs, long training time	Effective for critical tasks and large datasets; risk of overfitting for small data

6. ADAPTING THE SENTIMENT CLASSIFICATION MODEL FOR UNLABELED HOTEL REVIEWS

In the project, a supervised sentiment classification model (such as Sentence-BERT combined with a classifier) was trained on labeled datasets, where each review had a known sentiment label (positive or negative). However, in this proposed question, the hotel reviews consist only of raw text, without any rating scores or sentiment labels. This absence of labels introduces a significant challenge. The supervised model cannot be directly trained or evaluated on this data. Therefore, alternative unsupervised, semi-supervised, or transfer learning approaches are required to adapt the model.

6.1 Expected Performance of the Existing Model

If we apply the existing supervised sentiment classifier (trained on other domains like IMDb or product reviews) directly to hotel reviews, the performance will likely degrade due to:

1. **Domain Shift:** The model was trained on data with different vocabulary and expressions (e.g., movie reviews). Hotel reviews contain domain-specific terms like check-in, amenities, breakfast, and staff service, which may not exist in the training data.

2. **Contextual Mismatch:** Phrases such as “room was small but cozy” or “staff was late but helpful” can carry mixed sentiment, which the previous model may misinterpret.

3. **Lack of Fine-tuning:** Without retraining or adaptation, the model cannot adjust to the new linguistic patterns and tone of the hotel domain.

Thus, while the model may still produce some valid predictions due to its general language understanding, accuracy, recall, and precision will drop significantly unless modifications are made.

6.2 Proposed Modifications and Alternative Approaches

To handle the unlabeled hotel reviews effectively, several techniques can be applied. These methods combine unsupervised, semi-supervised, and transfer learning principles.

6.2.1 Unsupervised or Semi-supervised Learning

- **Approach:** Generate sentence embeddings using a pre-trained model like Sentence-BERT or MiniLML6-v2. Apply an unsupervised clustering algorithm such as K-Means or DBSCAN on these embeddings to group semantically similar reviews. After clustering, manually inspect a small number of reviews from each cluster to determine their general sentiment (e.g., positive, negative, neutral). Assign pseudo-labels to these clusters.

- **Why it works:** Sentence embeddings encode semantic similarity. Reviews expressing similar opinions (e.g., complaints or praise) will naturally fall into the same cluster even without explicit labels.

- **Modification:** Once pseudo-labels are assigned, a semi-supervised model can be trained using this partially labeled data, refining sentiment boundaries and improving performance.

6.2.2 Lexicon-based Sentiment Analysis

- **Approach:** Use predefined sentiment lexicons such as VADER, TextBlob, or SentiWordNet to analyze the polarity of words in each review. These lexicons assign sentiment scores to individual words, and the total review sentiment is calculated based on the sum of these scores.

- **Why it works:** Lexicon-based approaches do not require labeled training data. They work well for text that uses straightforward opinion words, such as "clean room", "rude staff", or "amazing service".

- **Modification:** Lexicons can be customized for the hotel domain by adding or adjusting hospitality-related terms (e.g., "check-in", "wifi speed", "breakfast quality", "front desk"). Assigning appropriate sentiment weights to these domain-specific terms can improve accuracy.

6.2.3 Zero-shot or Few-shot Classification Using Large Language Models

- **Approach:** Utilize zero-shot classification models (e.g., facebook/bart-large-mnli, deberta-v3-large-mnli, or GPT-based APIs). In this setup, no training is needed — instead, you define possible sentiment labels (e.g., "positive", "neutral", "negative"), and the model predicts which label best fits each review based on its pre-trained knowledge.

- **Why it works:** These large pre-trained models already understand sentiment and reasoning from massive text corpora. Hence, they can generalize sentiment prediction even in unseen domains.

- **Modification:** The performance can be improved using domain-specific prompts, such as: "This hotel review expresses a ___ sentiment." This helps guide the model to interpret the text in the correct context**.**

6.2.4 Weak Supervision / Self-training

- **Approach:** Start by generating pseudo-labels for the hotel reviews using lexicon-based or zero-shot methods. Then, use these pseudo-labeled reviews to fine-tune the existing SBERT + classifier pipeline. The model can be retrained iteratively, refining its predictions and learning from high-confidence pseudo-labels.

- **Why it works:** Weak supervision allows the model to learn from noisy but abundant pseudo labeled data. Over multiple iterations, the model becomes more accurate.

- **Modification:** Apply confidence filtering — only include pseudo-labeled examples where the model's confidence exceeds a threshold (e.g., 0.9). This reduces noise and improves reliability.

6.2.5 Transfer Learning from Similar Domains

- **Approach:** Use a pre-trained sentiment model (e.g., trained on IMDb, Yelp, or Amazon reviews) and perform domain adaptation on hotel reviews. The model's encoder (e.g., BERT layers) can be exposed to unlabeled hotel text to adjust its embeddings to hotel vocabulary and style.

- **Why it works:** Sentiment expressions (like "good", "bad", "excellent", "disappointing") are often shared across domains. Transfer learning helps reuse this prior knowledge efficiently.

- **Modification:** Fine-tune the model's embedding layer or language modeling head on hotel-specific text (unsupervised fine-tuning) before reusing it for sentiment

6.2.6 Human-in-the-loop Labelling

- **Approach:** Involve human annotators to label a small subset of hotel reviews (e.g., 500 1000 samples). Then, apply active learning, where the model selects uncertain samples for human review in subsequent iterations.

- **Why it works:** This process focuses human effort on the most informative samples, quickly improving the model's accuracy.

- **Modification:** Use SBERT embeddings to identify diverse examples for labeling, ensuring that labeled data covers different types of opinions and topics.

Method	Label Requirement	Learning Type	Strengths	Limitations/ Weaknesses
Unsupervised Clustering	None	Unsupervised	Finds natural sentiment groups	Requires manual cluster labeling
Lexicon-based	None	Zero-shot / Few-shot	Simple, interpretable; no training needed	Struggles with complex language; moderate accuracy
Semi-Supervised Transfer Learning	Minimal labels possible	Semi-Supervised / Transfer Learning	Improves supervision iteratively; adapts model to hotel domain	Noisy pseudo-labels; some manual effort required
Human-in-the-loop	Small subset	Active Learning	Efficient human labeling; domain adaptation	Needs computational resources; some manual effort required

In summary, a traditional supervised sentiment classifier cannot be directly applied to the unlabeled hotel review dataset. To achieve effective performance, it must be adapted using unsupervised, semi-supervised, transfer, or hybrid learning techniques.

Methods such as unsupervised clustering, lexicon-based analysis, and zero-shot classification allow initial exploration of sentiment without labels, while weak supervision, transfer learning, and human-in-the-loop labeling can progressively enhance model performance. Combining these strategies enables the model to understand hotel-domain sentiment expressions and deliver accurate sentiment predictions despite the absence of explicit rating scores.

7. HANDLING NOISY LABELS IN SENTIMENT CLASSIFICATION

The presence of noisy annotations fundamentally compromises the learning process of a supervised classification algorithm. When a supervised classification algorithm (like a Decision Tree, SVM, or Neural Network) is trained on sentiment data with inaccurate (noisy) rating scores, its performance is severely compromised.

7.1 Effects of training a classifier on Noisy Data

7.1.1 Impaired decision boundaries: The classifier's primary goal is to find a clear boundary that separates classes. With noisy data, samples are incorrectly placed in the training space (e.g., a clearly positive review is labelled 'Negative'). The algorithm must then twist and complicate its decision boundary to correctly classify these mislabelled outliers, leading to a highly complex and unstable model.

7.1.2 Overfitting to errors: High-capacity models, such as deep neural networks or complex decision trees, are powerful enough to not only learn the underlying true pattern but also to memorize the random noise. When a model overfits to noise, it achieves excellent performance on the flawed training set but performs poorly when deployed on a clean, real-world test set a massive decline in generalization ability.

7.1.3 Biased Loss Minimization: Standard training protocols minimize a loss function (like Cross-Entropy) that treats all misclassifications equally. A misclassification due to a true pattern is treated the same as a misclassification caused by a flipped label. The model expends significant effort optimizing its parameters to satisfy the noisy labels, which ultimately moves it away from the optimal configuration for the true, clean data distribution. The overall result is a model that is brittle and unreliable.

7.2 Approaches to Handling Noisy Data

Approaches to improve algorithm robustness and combat label noise, techniques focus on either preventing the noise from entering the learning process, mitigating its impact during training, or correcting the labels. The general methods are

7.2.1 Robust loss Functions (Mitigation)

Instead of modifying the data, this approach changes the objective function the model minimizes, making it less sensitive to outliers (the noisy samples). Standard practice: Sentiment models typically use Cross-Entropy Loss. CE loss applies a very large, non-linear penalty to confidently misclassified examples, causing the model to desperately try to correct a noisy sample it feels confident about.

- **Improvement:** switch to a loss function that scales the penalty linearly, such as Mean Absolute Error Loss or a hybrid loss like Generalized Cross-Entropy Loss.
- **Mechanism:** MAE Loss, for instance, prevents the loss value from growing unbounded. An extremely wrong, noisy label will only increase the total loss linearly, preventing the model from

spending too much optimization power on satisfying that single erroneous point. This effectively makes the training process more robust to outliers.

7.2.2 Sample Selection and Reweighting (Prevention):

This is a two-step process that uses the model itself to identify and ignore problematic training samples.

- Approach: Train the model for a few initial epochs, during which it tends to learn the general, clean patterns before it starts memorizing the noise. Then, use the model's output to assess the quality of the labels.

- Filtering: Identify samples where the loss value is very small. Small-loss samples are likely ones the model already learned correctly, meaning their labels are probably clean. Focus training on these "trusted" samples. Conversely, samples with extremely large loss may be discarded, as they are likely the hard-to-fit noisy outliers.

- Reweighting: Assign a dynamic weight to the loss calculated for each sample. If the model's predicted probability for the sample's assigned label is low, the weight omega is reduced, meaning that sample contributes less to the overall gradient update.

- Mechanism: This technique allows the model to prioritize learning from the data it deems most consistent with the patterns it has already established, effectively creating a cleaner training subset dynamically.

7.2.3 Noise transition modelling (Correction):

This is a more mathematically rigorous approach that attempts to model the error process itself. approach: estimate the noise transition matrix. This matrix is an N * N matrix (where N is the number of classes) where each entry Tij represents the probability that an original true label j was incorrectly flipped to the noisy label.

- **Improvement:** Once the transition matrix T is estimated (either pre-calculated or learned dynamically), the model can mathematically de-contaminate the loss function. The predicted probability distribution of the noisy data can be "uncorrupted" by multiplying it by the inverse of the transition matrix, T^{-1}.

- **Mechanism:** By learning the statistical properties of the noise, the algorithm can be trained on the entire dataset while correcting the bias introduced by the label errors, leading to a much more accurate estimate of the true classification loss.

7.3 Solutions in the context of learning neural networks.

When dealing with noisy labels in sentiment classification, neural networks are prone to overfitting, as they may try to memorize the incorrect annotations. To improve robustness and generalization, several strategies can be employed to limit model complexity and reduce sensitivity to noise:

7.3.1 Dropout

Dropout is a regularization technique where, during training, a fraction of neurons in each layer is randomly “dropped out” or ignored in each forward pass. This prevents the network from relying too

heavily on any single neuron or feature, forcing it to learn more distributed and robust representations. Dropout not only helps mitigate overfitting due to noise but also improves generalization on unseen data. Adjusting the dropout rate allows a balance between regularization and learning capacity.

7.3.2 Early Stopping

Early stopping monitors the model's performance on a validation set during training. Once the validation loss stops improving or begins to increase (indicating overfitting), training is halted, and the model reverts to the weights corresponding to the best validation performance. This prevents the network from overfitting noisy labels by stopping before it fully memorizes the noise. It is a simple yet highly effective technique for controlling overfitting in practice.

7.3.3 Weight Sharing

Weight sharing constrains multiple neurons to use the same set of parameters. This reduces the number of free parameters in the network, thereby limiting its capacity to overfit noisy data. It is commonly used in architectures like convolutional neural networks, where the same filter is applied across different positions, but can also be adapted in other network designs to enforce structured regularization.

7.3.4 Dimensionality Reduction

Reducing the dimensionality of the input data or embeddings can help simplify the learning task. Techniques like PCA, autoencoders, or selecting a subset of informative features reduce noise in the input representation and prevent the network from overfitting irrelevant or noisy features. Lower-dimensional data typically results in faster training and better generalization.

7.3.5 Label Smoothing

Label smoothing replaces hard one-hot labels with slightly softened probabilities (e.g., 0.9 for the correct class, 0.1 distributed among others). This discourages the network from becoming overconfident on potentially noisy labels, reducing the risk of overfitting and improving generalization.

Incorporating these strategies ensures that neural networks are less sensitive to noisy or inaccurate labels in sentiment classification. Techniques like dropout, early stopping, weight sharing, and dimensionality reduction reduce model complexity and overfitting, allowing the network to focus on meaningful patterns rather than memorizing noise. When combined thoughtfully, these approaches significantly improve robustness, especially in scenarios with imperfect annotations.

8. REFERENCES

[1] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs Up? Sentiment Classification Using Machine Learning Techniques," *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 79–86, 2002.

[2] P. D. Turney, "Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews," *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 417–424, 2002.

[3] B. Liu, *Sentiment Analysis and Opinion Mining*, *Synthesis Lectures on Human Language Technologies*, vol. 5, no. 1, pp. 1–167, 2012.

[4] E. Cambria, D. Das, S. Bandyopadhyay, and A. Feraco (Eds.), *A Practical Guide to Sentiment Analysis*. Springer, 2017.

[5] X. Glorot, A. Bordes, and Y. Bengio, "Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach," *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pp. 513–520, 2011.

[6] Y. Ziser and R. Reichart, "Deep Unsupervised Domain Adaptation for Text Classification Using Domain-Invariant Features," *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 321–330, 2018.

[7] M. Pontiki, D. Galanis, H. Papageorgiou, I. Androutsopoulos, S. Manandhar, M. Al-Smadi, A. Mohammad, B. Kiritchenko, L. M. Zhang, and S. Zhu, "SemEval-2014 Task 4: Aspect-Based Sentiment Analysis," *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval)*, pp. 27–35, 2014.

[8] H. Xu, B. Liu, L. Shu, and P. S. Yu, "BERT Post-Training for Review Reading Comprehension and Aspect-Based Sentiment Analysis," *Proceedings of NAACL-HLT*, pp. 2324–2335, 2019.

[9] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, "Unsupervised Cross-lingual Representation Learning at Scale," *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 8440–8451, 2020.

[10] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pp. 4171–4186, 2019.

[11] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks," *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

[12] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter," *arXiv preprint* arXiv:1910.01108, 2019.

[13] E. Cambria, "Affective Computing and Sentiment Analysis," *IEEE Intelligent Systems*, vol. 31, no. 2, pp. 102–107, 2016.

[14] B. Liu and L. Zhang, “A Survey of Opinion Mining and Sentiment Analysis,” in *Mining Text Data*, C. C. Aggarwal and C. Zhai, Eds. Springer, pp. 415–463, 2012.

[15] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” *arXiv preprint* arXiv:1301.3781, 2013.

[16] J. Pennington, R. Socher, and C. D. Manning, “GloVe: Global Vectors for Word Representation,” *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[17] Y. Kim, “Convolutional Neural Networks for Sentence Classification,” *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1746–1751, 2014.

[18] D. Tang, B. Qin, and T. Liu, “Document Modeling with Gated Recurrent Neural Network for Sentiment Classification,” *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1422–1432, 2015.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention Is All You Need,” *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 5998–6008, 2017.

[20] D. Cer, Y. Yang, S. Kong, N. Hua, N. Limtiaco, R. St. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, B. Strope, and R. Kurzweil, “Universal Sentence Encoder,” *arXiv preprint* arXiv:1803.11175, 2018.

[21] J. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, “TinyBERT: Distilling BERT for Natural Language Understanding,” *arXiv preprint* arXiv:1909.10351, 2019.

[22] P. Ruder, I. Vulić, and S. Ruder, “A Survey of Cross-Lingual Word Embedding Models,” *Journal of Artificial Intelligence Research*, vol. 65, pp. 569–631, 2019.