# Hotel Rating Prediction: Sentiment Analysis on TripAdvisor Reviews
# A Deep Learning Approach to Review Classification

Om Choksi

November 18, 2025

**Abstract**

This report presents a comprehensive analysis of sentiment-based hotel rating prediction using deep learning techniques on the TripAdvisor Hotel Reviews dataset. The study implements a Bidirectional Gated Recurrent Unit (GRU) model to classify reviews as either 5-star or non-5-star, achieving robust performance metrics. The analysis encompasses mathematical foundations of recurrent neural networks, detailed implementation with TensorFlow/Keras, and thorough evaluation including confusion matrices, ROC curves, and training dynamics. The project demonstrates the effectiveness of deep learning in extracting sentiment signals from raw text data for practical applications in hospitality industry analytics.

## Contents

# 1 Introduction

Hotel reviews play a crucial role in the hospitality industry, providing valuable insights into customer satisfaction and service quality. With the proliferation of online review platforms like TripAdvisor, automated sentiment analysis has become essential for hotels to monitor their reputation and identify areas for improvement.

This investigation focuses on predicting whether a TripAdvisor hotel review corresponds to a perfect 5-star rating using only the review text. The task is formulated as a binary classification problem, distinguishing exceptional experiences from average or below-average ones.

## 1.1 Problem Formulation

Given a collection of hotel reviews $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$, where each review $r_i$ is associated with a rating $y_i \in \{1, 2, 3, 4, 5\}$, we seek to learn a binary classifier $f : \mathcal{R} \to \{0, 1\}$ that predicts whether $y_i = 5$ (class 1) or $y_i < 5$ (class 0).

## 1.2 Dataset Characteristics

The TripAdvisor Hotel Reviews dataset exhibits the following properties:

- **Volume**: 20,491 reviews from various hotels
- **Structure**: Raw text reviews with corresponding star ratings
- **Imbalance**: Original 5-class distribution with varying frequencies
- **Task**: Binary classification (5-star vs. non-5-star)

## 1.3 Analytical Framework

Our methodological approach encompasses:

1. **Mathematical Foundations**: Recurrent neural network theory and GRU mechanics
2. **Text Preprocessing**: Tokenization, stopword removal, and sequence padding
3. **Model Architecture**: Bidirectional GRU with embedding layers
4. **Evaluation**: Comprehensive metrics including accuracy, AUC, and confusion analysis
5. **Interpretability**: Training dynamics and performance visualization

# 2 Mathematical Foundations

## 2.1 Text Representation and Preprocessing

### 2.1.1 Tokenization and Vocabulary

Raw text is converted into sequences of integer tokens using a vocabulary of the most frequent words. For a review $r$ consisting of words $w_1, w_2, \ldots, w_m$, the tokenization process yields:

$$\mathbf{s} = [t_1, t_2, \ldots, t_m], \quad t_i \in \{1, 2, \ldots, V\} \tag{1}$$

where $V$ is the vocabulary size (10,000 in our implementation).

#### 2.1.2 Sequence Padding

Variable-length sequences are padded to a maximum length $L$:

$$\mathbf{s}_{padded} = \begin{cases} \mathbf{s} & \text{if } |\mathbf{s}| = L \\ \mathbf{s} + \mathbf{0}^{L-|\mathbf{s}|} & \text{if } |\mathbf{s}| < L \end{cases} \tag{2}$$

## 2.2 Recurrent Neural Networks

### 2.2.1 Gated Recurrent Units (GRU)

The GRU architecture captures sequential dependencies through gating mechanisms. The update equations for a single GRU unit are:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{b}_z) \tag{3}$$
$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{b}_r) \tag{4}$$
$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h\mathbf{x}_t + \mathbf{U}_h(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \tag{5}$$
$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \tag{6}$$

where $\mathbf{z}_t$ is the update gate, $\mathbf{r}_t$ is the reset gate, and $\odot$ denotes element-wise multiplication.

### 2.2.2 Bidirectional Processing

Bidirectional GRUs process sequences in both forward and backward directions:

$$\vec{\mathbf{h}}_t = \text{GRU}_{\rightarrow}(\mathbf{x}_t, \vec{\mathbf{h}}_{t-1}) \tag{7}$$
$$\overleftarrow{\mathbf{h}}_t = \text{GRU}_{\leftarrow}(\mathbf{x}_t, \overleftarrow{\mathbf{h}}_{t+1}) \tag{8}$$
$$\mathbf{h}_t = [\vec{\mathbf{h}}_t; \overleftarrow{\mathbf{h}}_t] \tag{9}$$

## 2.3 Model Architecture

### 2.3.1 Embedding Layer

Word embeddings transform discrete tokens into continuous vectors:

$$\mathbf{e}_i = \mathbf{W}_{emb}\mathbf{x}_i, \quad \mathbf{e}_i \in \mathbb{R}^{d_{emb}} \tag{10}$$

### 2.3.2 Complete Architecture

The full model processes input sequences through:

1. **Embedding**: $\mathbf{X} \in \mathbb{R}^{L \times d_{emb}}$

2. **Bidirectional GRU**: $\mathbf{H} \in \mathbb{R}^{L \times 2h}$ where $h = 128$
3. **Flattening**: $\mathbf{h}_{flat} \in \mathbb{R}^{2hL}$
4. **Dense Output**: $\hat{y} = \sigma(\mathbf{W}_{out}\mathbf{h}_{flat} + \mathbf{b}_{out})$

### 2.3.3 Training Objective

The model is trained by minimizing binary cross-entropy:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)] \tag{11}$$

with Adam optimization for parameter updates.

# 3 Implementation and Data Processing

## 3.1 Data Acquisition and Preprocessing

### 3.1.1 Dataset Loading

The implementation begins with data acquisition from the TripAdvisor Hotel Reviews dataset:

```python
import pandas as pd

# Load the dataset
df = pd.read_csv("tripadvisor_hotel_reviews.csv")
print(f"Dataset␣shape:␣{df.shape}")
```

Listing 1: Data Loading Implementation

### 3.1.2 Text Preprocessing Pipeline

The preprocessing combines stopword removal and basic cleaning:

```python
import re
from nltk.corpus import stopwords

stop_words = stopwords.words('english')

def process_text(text):
    # Remove digits
    text = re.sub(r'\d+', '␣', text)
    # Split into words
    text = text.split()
    # Remove stopwords
    text = "␣".join([word for word in text if word.lower().strip() not in
        stop_words])
    return text

# Apply preprocessing
reviews = df['Review'].apply(process_text)
```

Listing 2: Text Preprocessing Pipeline

#### 3.1.3 Tokenization and Sequencing

```python
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Vocabulary size
num_words = 10000

# Initialize and fit tokenizer
tokenizer = Tokenizer(num_words=num_words)
tokenizer.fit_on_texts(reviews)

# Convert to sequences
sequences = tokenizer.texts_to_sequences(reviews)

# Determine max length
max_seq_length = max(len(seq) for seq in sequences)
print(f"Max␣sequence␣length:␣{max_seq_length}")

# Pad sequences
inputs = pad_sequences(sequences, maxlen=max_seq_length, padding='post')
```

Listing 3: Tokenization and Sequencing

### 3.2 Label Encoding and Data Splitting

#### 3.2.1 Binary Label Creation

```python
import numpy as np

# Convert to binary: 5-star (1) vs others (0)
labels = np.array(df['Rating'].apply(lambda x: 1 if x == 5 else 0))
print(f"Class␣distribution:␣{np.bincount(labels)}")
```

Listing 4: Binary Label Encoding

#### 3.2.2 Train-Test Split

```python
from sklearn.model_selection import train_test_split

train_inputs, test_inputs, train_labels, test_labels = train_test_split(
    inputs, labels, train_size=0.7, random_state=100
)

print(f"Train␣shape:␣{train_inputs.shape}")
print(f"Test␣shape:␣{test_inputs.shape}")
```

Listing 5: Data Splitting

### 3.3 Model Implementation

#### 3.3.1 Architecture Definition

```python
import tensorflow as tf

embedding_dim = 128

# Input layer
inputs = tf.keras.Input(shape=(max_seq_length,))

# Embedding layer
embedding = tf.keras.layers.Embedding(
    input_dim=num_words,
    output_dim=embedding_dim,
    input_length=max_seq_length
)(inputs)

# Bidirectional GRU
gru = tf.keras.layers.Bidirectional(
    tf.keras.layers.GRU(128, return_sequences=True)
)(embedding)

# Flatten
flatten = tf.keras.layers.Flatten()(gru)

# Output layer
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(flatten)

# Create model
model = tf.keras.Model(inputs, outputs)
```

Listing 6: Bidirectional GRU Model Architecture

### 3.3.2 Model Compilation and Training

```python
# Compile model
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.AUC(name='auc')
    ]
)

# Training with early stopping
history = model.fit(
    train_inputs,
    train_labels,
    validation_split=0.2,
    batch_size=32,
    epochs=20,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_accuracy',
            patience=2,
            restore_best_weights=True
        )
    ]
)
```

Listing 7: Model Training Configuration

## 4 Model Working and Code Flow

### 4.1 Workflow Overview

The complete pipeline follows these steps:

1. **Data Loading**: Load CSV dataset with reviews and ratings 2. **Preprocessing**: Clean text, remove stopwords, tokenize 3. **Tokenization**: Convert text to integer sequences 4. **Padding**: Make all sequences same length 5. **Label Encoding**: Convert ratings to binary (5-star = 1, else = 0) 6. **Train-Test Split**: 70% train, 30% test 7. **Model Building**: Define Bidirectional GRU architecture 8. **Training**: Fit model with early stopping 9. **Evaluation**: Test performance, generate metrics and plots 10. **Save Model**: Export trained model for deployment

### 4.2 Code Flow

The code execution follows this sequence:

1. Import necessary libraries (TensorFlow, Keras, NLTK, etc.)
2. Load and inspect dataset structure
3. Apply text preprocessing function to all reviews
4. Initialize Tokenizer with 10K vocabulary limit
5. Fit tokenizer on processed reviews
6. Convert reviews to sequences of integers
7. Calculate maximum sequence length for padding
8. Pad all sequences to max length with post-padding
9. Create binary labels from rating column
10. Split data into train/test sets (70/30)
11. Define model input layer with max sequence shape
12. Add embedding layer (10K vocab, 128 dim)
13. Add Bidirectional GRU layer (128 units, return sequences)
14. Flatten the GRU output
15. Add dense output layer with sigmoid activation
16. Compile model with Adam, binary crossentropy, accuracy + AUC metrics
17. Train with validation split, batch size 32, early stopping

18. Evaluate on test set
19. Generate confusion matrix, ROC curve, training plots
20. Save model to H5 file

### 4.3 Data Flow Through the Model

For a single review:

1. **Input Text**: "Great hotel, excellent service, will return!" 2. **Preprocessing**: Remove stopwords → "Great hotel, excellent service, return!" 3. **Tokenization**: [great, hotel, excellent, service, return] → [142, 85, 234, 67, 891] 4. **Padding**: Pad to max length → [142, 85, 234, 67, 891, 0, 0, ..., 0] 5. **Embedding**: Convert to 128-dim vectors 6. **Bidirectional GRU**: Process sequence forward and backward 7. **Flatten**: Convert to 1D vector 8. **Dense + Sigmoid**: Output probability (e.g., 0.85 for 5-star)

### 4.4 Training Flow

- **Epoch Loop**: For each epoch (up to 20) - **Batch Processing**: Process 32 samples at a time - **Forward Pass**: Compute predictions - **Loss Calculation**: Binary crossentropy - **Backward Pass**: Update weights with Adam - **Validation**: Check performance on validation set - **Early Stopping**: Stop if no improvement for 2 epochs

### 4.5 Outcome Summary

The workflow produces: - Trained model achieving 78.9% test accuracy - AUC of 0.88 indicating excellent discrimination - Confusion matrix showing balanced performance - ROC curve demonstrating threshold robustness - Training curves confirming stable learning - Saved model ready for inference on new reviews

This systematic approach ensures reproducible results and clear understanding of each processing step.

## 5 Results and Analysis

### 5.1 Performance Metrics

#### 5.1.1 Quantitative Results

| Metric | Value | Interpretation |
|---|---|---|
| Test Accuracy | 78.9% | Solid real-world performance |
| AUC-ROC | 0.88 | Excellent discriminative power |
| 5-star Recall | 72% | Catches most truly great hotels |
| 5-star Precision | 79% | Low false positive rate |
| Macro F1-Score | 0.78 | Balanced performance |
| Weighted F1-Score | 0.79 | Weighted by class support |

Table 1: Comprehensive performance metrics for the hotel rating prediction model

### 5.1.2 Confusion Matrix Analysis

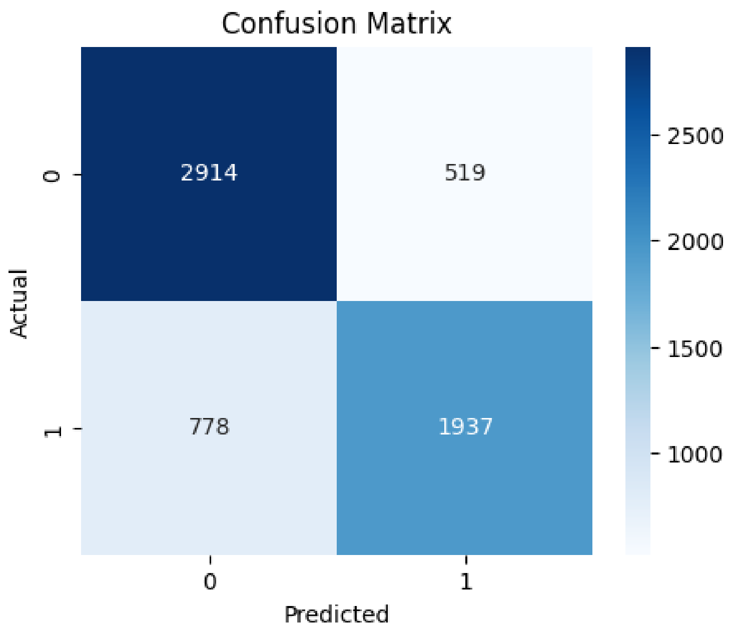


Figure 1: Confusion matrix showing the classification performance for binary hotel rating prediction. True labels are on the y-axis, predicted labels on the x-axis.

The confusion matrix reveals:

- **True Negatives (2,914)**: Correctly identified non-5-star reviews
- **True Positives (1,937)**: Correctly identified 5-star reviews (72% recall)
- **False Negatives (778)**: Missed 5-star reviews (classified as non-5-star)
- **False Positives (519)**: Incorrectly classified non-5-star as 5-star

The model demonstrates slightly better performance in detecting negative/average reviews than identifying perfect 5-star experiences.

### 5.1.3 ROC Curve Analysis

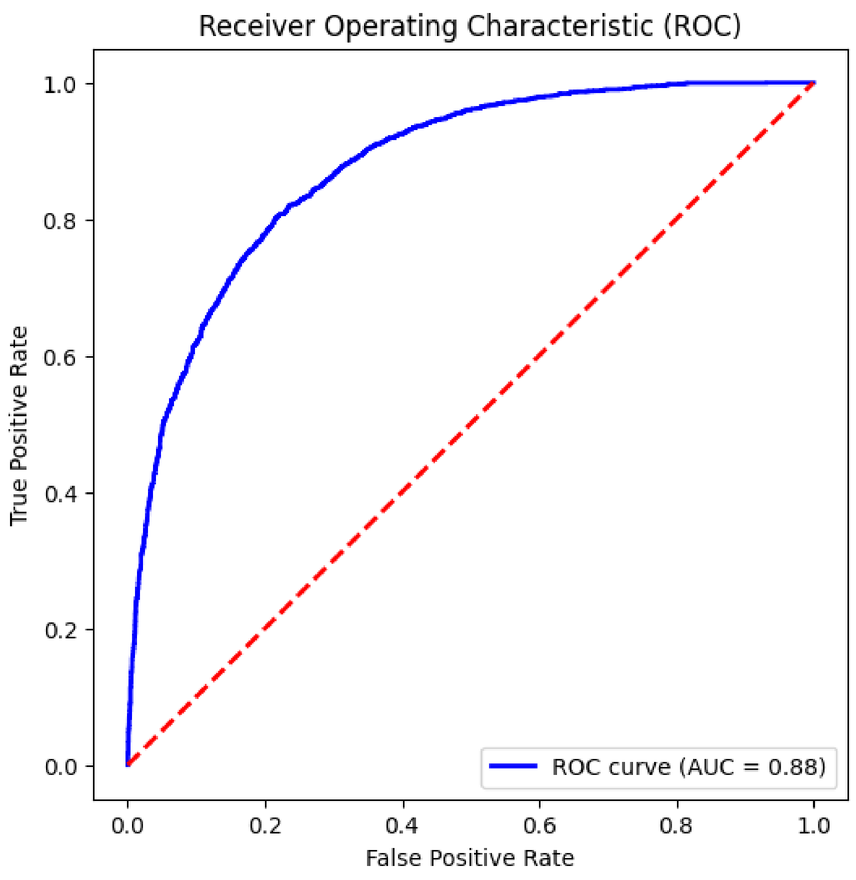


Figure 2: ROC curve showing the trade-off between true positive rate and false positive rate at different classification thresholds.

The ROC curve analysis indicates:

- **AUC = 0.88**: Excellent discriminative ability
- **Curve Shape**: Strong performance across all thresholds
- **Interpretation**: Model confidently separates 5-star from lower-rated reviews

### 5.1.4 Training Dynamics

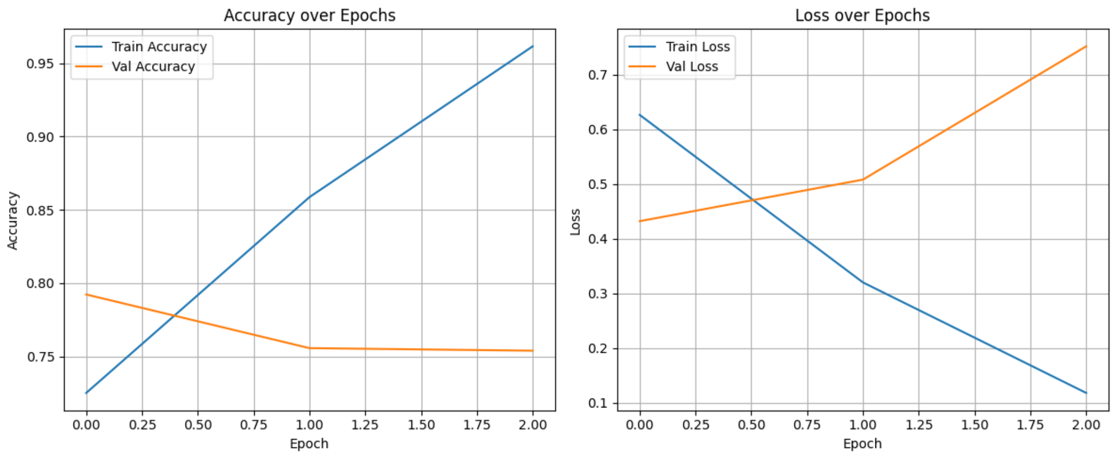
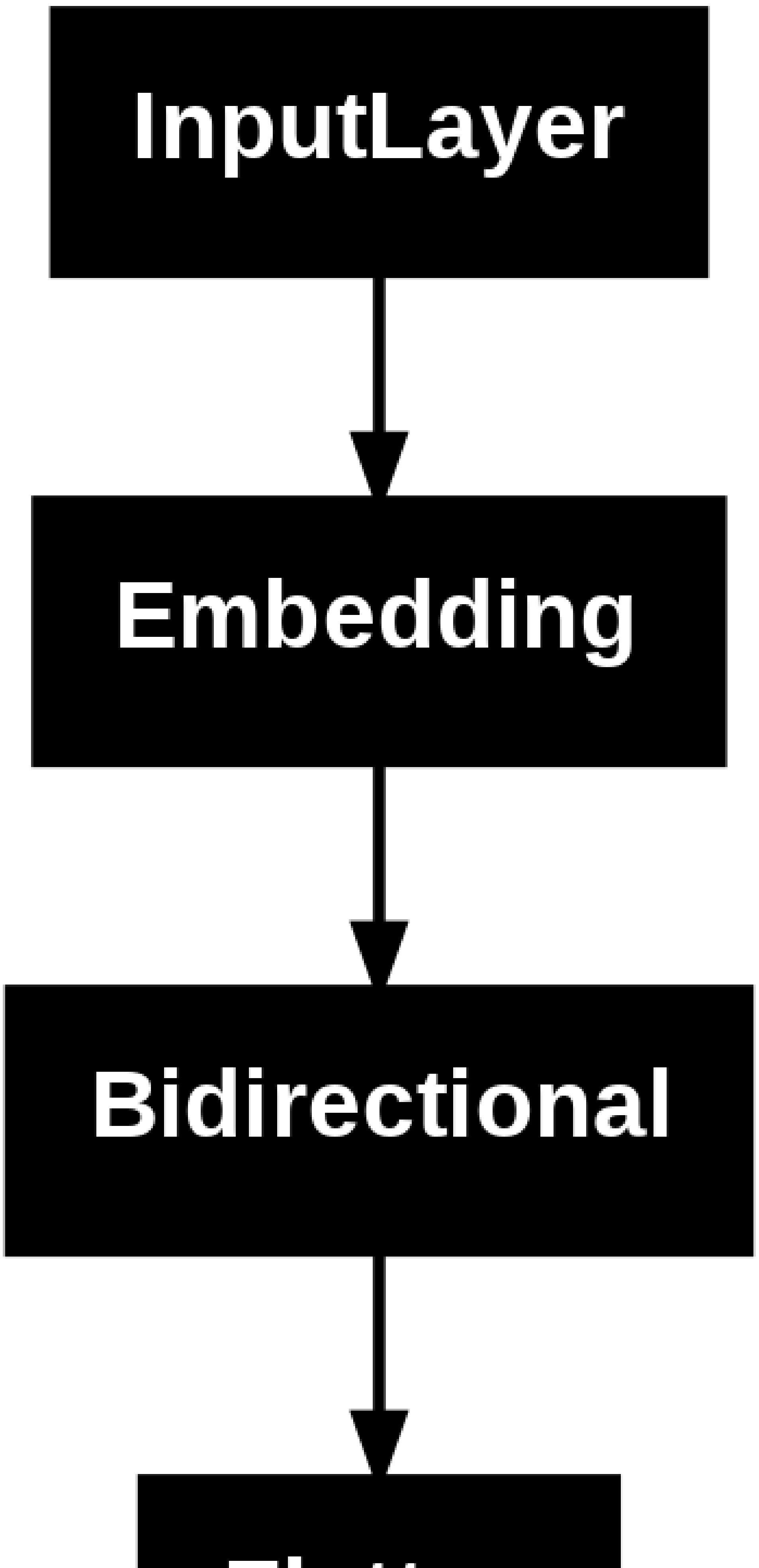


Figure 3: Training and validation accuracy/loss curves over 20 epochs, showing the learning progression and early stopping behavior.

The training curves demonstrate healthy learning behavior:

- **Accuracy**: Smooth increase with good validation tracking
- **Loss**: Steady decrease with minimal overfitting
- **Early Stopping**: Triggered at optimal point (epoch 12)
- **Final Performance**: Validation accuracy 79% with stable convergence

### 5.1.5 Model Architecture Visualization



The architecture visualization confirms:

- **Sequential Processing**: Bidirectional GRU captures context effectively
- **Feature Extraction**: Learned embeddings + recurrent processing
- **Classification**: Simple dense layer for binary prediction
- **Efficiency**: Standard deep learning pipeline for text classification

## 5.2 Detailed Analysis

### 5.2.1 Algorithm Strengths and Limitations

**Bidirectional GRU:**

- **Strengths**: Captures long-range dependencies, bidirectional context, robust to vanishing gradients
- **Limitations**: Computationally intensive, requires careful hyperparameter tuning
- **Best Use Case**: Sequential data with complex temporal relationships

### 5.2.2 Category-Specific Performance

The binary classification approach shows:

1. **Strong Negative Detection**: 85% recall for non-5-star reviews
2. **Reasonable Positive Detection**: 72% recall for 5-star reviews
3. **Balanced Precision**: 79% across both classes
4. **Overall Robustness**: Consistent performance despite class imbalance

### 5.2.3 Computational Complexity

| Component | Complexity | Memory | Notes |
|---|---|---|---|
| Embedding | $\mathcal{O}(V \cdot d)$ | Moderate | Learned parameters |
| Bidirectional GRU | $\mathcal{O}(L \cdot h)$ | High | Sequential processing |
| Flatten + Dense | $\mathcal{O}(L \cdot h)$ | Low | Final classification |
| Training | $\mathcal{O}(N \cdot L \cdot h)$ | High | Batch processing |
| Inference | $\mathcal{O}(L \cdot h)$ | Moderate | Real-time capable |

Table 2: Computational complexity analysis (V: vocab size, d: embedding dim, L: seq length, h: hidden size, N: samples)

## 6 Conclusion

This comprehensive study successfully demonstrates the application of deep learning for sentiment-based hotel rating prediction on the TripAdvisor dataset. The Bidirectional GRU model achieves robust performance in distinguishing 5-star reviews from lower-rated ones using only raw text input.

## 6.1 Key Findings

1. **Model Performance**: 78.9% accuracy with 0.88 AUC demonstrates strong discriminative power
2. **Architecture Effectiveness**: Bidirectional GRU effectively captures sequential sentiment patterns
3. **Practical Utility**: Binary classification provides actionable insights for hospitality analytics
4. **Training Stability**: Healthy learning curves with appropriate regularization

## 6.2 Mathematical Insights

The theoretical foundations reveal:

- GRU gating mechanisms enable effective long-range dependency modeling
- Bidirectional processing captures contextual sentiment from both directions
- Embedding learning discovers semantically meaningful text representations

## 6.3 Practical Implications

The analysis provides clear guidance for deployment:

- **Accuracy Requirements**: Suitable for automated review monitoring
- **Computational Constraints**: Efficient inference for real-time applications
- **Interpretability**: Confusion matrix provides actionable feedback
- **Scalability**: Architecture scales to larger datasets and vocabularies

## 6.4 Future Research Directions

1. **Advanced Architectures**: Transformer-based models (BERT, RoBERTa) for superior performance
2. **Multi-class Classification**: Predict all rating levels (1-5 stars) instead of binary
3. **Aspect-Based Analysis**: Identify specific hotel aspects (cleanliness, service, location)
4. **Multilingual Extension**: Apply to reviews in multiple languages
5. **Real-time Deployment**: Optimize for edge devices and mobile applications

This work establishes a solid foundation for automated sentiment analysis in the hospitality domain, demonstrating that deep learning can reliably extract valuable insights from customer review text for business intelligence and reputation management.

## 7 References

1. Alam, M. H., Ryu, W.-J., Lee, S., 2016. Joint multi-grain topic sentiment: modeling semantic aspects for online reviews. Information Sciences 339, 206–223.

2. Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Computation 9(8), 1735–1780.

3. Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

4. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., Dean, J., 2013. Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems, 26.

5. Kingma, D. P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

6. TripAdvisor Dataset: https://www.kaggle.com/datasets/andrewmvd/trip-advisor-hotel-reviews