Comparative Analysis of RNN Architectures for Sentiment Classification

1. Introduction

Sentiment Classification is a core Natural Language Processing (NLP) task that involves categorizing the emotional tone of a piece of text—such as a movie review, tweet, or product comment—into classes like **positive** or **negative**.

In this project, you will implement and evaluate multiple **Recurrent Neural Network (RNN)** architectures for sentiment classification, treating it as a **sequence classification** problem.

---

2. Dataset Selection

Dataset: [IMDb Movie Review Dataset (50,000 reviews)]

Dataset Preparation Guidelines

- Use the **predefined 50/50 split** (25k for training, 25k for testing).
- Preprocess the text as follows:
  - Lowercase all text.
  - Remove punctuation and special characters.
  - Tokenize sentences (use `Keras Tokenizer` or `nltk.word_tokenize`).
  - Keep only the **top 10,000 most frequent words**.
  - Convert each review to a sequence of token IDs.
  - **Pad or truncate** sequences to fixed lengths of **25, 50, and 100** words (you will test these variations).

---

3. Model Architecture

You will experiment with the following model configurations:

Category	Variations to Test
Architecture	RNN, LSTM, Bidirectional LSTM
Activation Function	Sigmoid, ReLU, Tanh
Optimizer	Adam, Stochastic Gradient Descent (SGD), RMSProp
Sequence Length	25, 50, 100
Stability Strategy	No strategy vs. Gradient Clipping

Model Design Notes

- Include an **embedding layer** (size: 100).
- Use **2 hidden layers** (hidden size: 64).
- Use **dropout** (0.3–0.5) to reduce overfitting.
- Batch size: 32.
- Use a **fully connected output layer** with a sigmoid activation for binary classification.
- Use **binary cross-entropy loss**.
- Fix all other hyperparameters when varying one factor (e.g., only change the optimizer, keep architecture and sequence length fixed).

---

4. Evaluation Experiments

You must systematically evaluate the effects of the variations listed above.
Measure performance using:

- **Accuracy**
- **F1-score (macro)**
- **Training time per epoch (seconds)**

Reporting Requirements

Create a summary table like:

Model	Activation	Optimizer	Seq Length	Grad Clipping	Accuracy	F1	Epoch Time (s)
RNN	ReLU	Adam	50	Yes	0.87	0.85	42.1

Plots of:

- Accuracy/F1 vs. Sequence Length
- Training Loss vs. Epochs (for best and worst models)

Reproducibility

Fix random seeds:

```
import torch, random, numpy as np
torch.manual_seed(42)
np.random.seed(42)
random.seed(42)
```

Report the hardware used (e.g., CPU only, RAM size).

---

5. Deliverables

5.1 Code Repository

A **well-structured GitHub repository** containing:

```
├── data/
├── src/
│   ├── preprocess.py
│   ├── models.py
│   ├── train.py
│   ├── evaluate.py
│   └── utils.py
├── results/
```

```
│   ├── metrics.csv
│   └── plots/
├── report.pdf
├── requirements.txt
└── README.md
```

5.2 README.md

Include:

- Setup instructions (Python version, dependencies)
  How to run training and evaluation scripts
- Expected runtime and output files

5.3 Project Report (PDF)

Your report should include:

1. **Dataset Summary:** Description of preprocessing and statistics (avg. review length, vocab size).
2. **Model Configuration:** Parameters (embedding dim, hidden size, number of layers, dropout, optimizer settings).
3. **Comparative Analysis:** Tables and charts comparing Accuracy, F1, and training time across experimental variations.
4. **Discussion:**
   - Which configuration performed best?
   - How did sequence length or optimizer affect performance?
   - How did gradient clipping impact stability?
5. **Conclusion:** Identify the **optimal configuration** under CPU constraints and justify your choice.

6. Evaluation Criteria

Code Implementation (25 points): The code runs correctly, is well-organized, and includes all key components — data preprocessing, model implementation, training, and evaluation.

Experimental Design (20 points): The project systematically tests the required variations (RNN, LSTM, Bidirectional LSTM, activation functions, optimizers, sequence lengths, and gradient clipping) using a controlled approach.

Results and Analysis (25 points): The report presents clear and accurate results, including accuracy, F1-score, and training time. Comparisons are well explained, and the best-performing configuration is identified with justification.

Report Quality (20 points): The written report is clear, well-structured, and self-contained. It explains the dataset, preprocessing, model setup, experiments, and conclusions in a logical manner.

Reproducibility and Documentation (10 points): The submission includes a README file, dependency list, and instructions so that others can easily reproduce the results.