Text and Sequence Data

This assignment explores Recurrent Neural Networks (RNNs) and their potential for handling and analyzing text-based sequence data, with a particular emphasis on sentiment analysis. The project centers on the IMDB movie review dataset, a widely used benchmark in natural language processing, to understand how RNN models perform under limited-data conditions and how embedding techniques can enhance efficiency.

Throughout this assignment, we will experiment with various model configurations and evaluate their performance. This includes examining the effects of data reduction, constrained training samples, and the use of both learned and pre-trained embeddings. The hands-on work reinforces key course concepts by illustrating the foundational ideas of deep learning, sequence modeling, and neural network design—especially in the context of optimizing performance when data is scarce.

Problem Statement
Text data is inherently sequential and highly dimensional, making it a poor fit for traditional machine learning approaches that treat inputs independently. Such models often struggle with tasks like sentiment analysis because they cannot capture context or temporal relationships within text sequences.

Recurrent Neural Networks (RNNs) were designed to address these limitations by modeling sequential patterns over time. This project leverages RNNs to classify the sentiment of movie reviews from the IMDB dataset, specifically under constrained data conditions.

The approach includes several strategies:

1. Limiting each review to a maximum of 150 words.
2. Training on a reduced dataset of only 100 samples.
3. Validating performance using 10,000 samples.
4. Restricting the vocabulary to the 10,000 most frequent words.
5. Comparing two model setups:
   a. a standard learned embedding layer, and
   b. a pre-trained word embedding layer—both followed by a Bidirectional RNN.

Primary Objectives
This project focuses on implementing RNNs for sentiment analysis on the IMDB dataset, with particular emphasis on low-resource scenarios. The goal is to build a model capable

of learning effectively even with very limited training data, while also examining how different embedding strategies affect accuracy and robustness.

The assignment aims to:

- Understand how RNNs capture and learn sequential dependencies in text.
- Compare the performance of learned embeddings versus pre-trained embeddings.
- Identify the minimum amount of training data needed to achieve reliable sentiment classification.

Ultimately, the project seeks to determine which preprocessing strategies and model configurations yield the best predictive performance when both data availability and computational resources are restricted.

Data Preparation

The dataset used is the IMDB movie review corpus, which contains 50,000 labeled reviews categorized as positive or negative. To simulate low-resource conditions and test model robustness with minimal processing overhead, several preprocessing steps were applied:

- **Truncation:** Each review is limited to 150 words to ensure uniform input length and reduce computation.
- **Vocabulary Limitation:** The vocabulary is restricted to the 10,000 most frequent words to control model complexity and minimize overfitting.
- **Sample Constraints:** Only 100 reviews are used for training, while 10,000 are reserved for validation.
- **Tokenization and Padding:** Reviews are tokenized into integer sequences and padded to maintain consistent input size.

These preprocessing choices are essential for improving computational efficiency and for assessing how well the RNN performs under constrained data conditions.

Model Architecture

The core architecture used in this project is a Bidirectional Recurrent Neural Network (BiRNN), which processes text sequences from both the forward and backward directions. This dual perspective helps the model capture richer contextual information—especially important in sentiment analysis, where the meaning of a word often depends heavily on the surrounding text.

Two separate model setups were implemented and evaluated:

1. **Embedding Model:**

   An embedding is a numerical representation of data, typically in the form of a vector, that captures the semantic or structural meaning of the input. In the context of text, embeddings convert words, sentences, or documents into high-dimensional vectors where similar items are positioned closer together in vector space. This allows algorithms to perform tasks such as similarity searches, clustering, and classification efficiently, because the relationships between data points are preserved in a way that computers can easily process."

   - **Embedding Layer:** This version learns word embeddings directly during training, allowing the model to develop task-specific vector representations.
   - **Bidirectional LSTM:** A bidirectional LSTM layer reads the sequence from both ends, enabling the model to incorporate contextual cues from preceding and following words.

2. **Pre-trained Embedding Model:**

   "A pretrained embedding model is a machine learning model that has already been trained on a large dataset to generate embeddings for text, images, or other data types. By leveraging knowledge learned from vast corpora, these models can produce high-quality vector representations for new data without the need to train from scratch. Pretrained embeddings are commonly used in natural language processing tasks such as semantic search, question answering, and recommendation systems, enabling developers to save time and achieve robust performance with minimal custom training."

   - Pre-trained Embedding Layer: Initialized with externally trained word vectors (e.g., GloVe), providing the model with richer semantic information from the start.
   - Bidirectional LSTM: Uses the same bidirectional LSTM architecture as the learned-embedding model to allow direct performance comparison.
   - Dense Output Layer: A fully connected layer responsible for producing the final sentiment classification (positive or negative).

Models Built	**Training Sample Size**	**Validation Size**	**Testing Sample Size**	**Model Type**	**Test Accuracy (%)**	**Test Loss**
Model 1	**100**	**10000**	**6000**	**Embedded Layer**	**51.3**	**0.693**
Model 2	**10000**	**10000**	**6000**	**Embedded Layer**	**50.2**	**0.693**
Model 3	**16000**	**10000**	**6000**	**Embedded Layer**	**89.3**	**0.255**
Model 4	**10000**	**10000**	**6000**	**One-Hot**	**87.9**	**0.308**
Model 5	**16000**	**10000**	**6000**	**LSTM Using Embedded Layer**	**82.98**	**0.80**
Model 6	**32000**	**10000**	**6000**	**LSTM Using Embedded Layer**	**95.1**	**0.184**
Model 7	**10000**	**10000**	**6000**	**Masking Enable**	**49.9**	**0.694**
Model 8	**100**	**10000**	**6000**	**Pretrained using GloVe**	**59.6**	**0.682**
Model 9	**15000**	**10000**	**6000**	**Pretrained with 4 LSTM hidden layers**	**67.2**	**0.572**
Model 10	**30000**	**10000**	**6000**	**Pretrained with 2 LSTM hidden layers**	**88.1**	**0.473**

1. Summary of Results

Ten models were trained and evaluated using different combinations of training sample sizes, input representations, and neural network architectures, including basic embedding layers, one-hot encoding, LSTM networks, masking approaches, and pretrained embeddings. Test accuracy and test loss were used to assess model performance across a fixed validation and testing set.

Performance ranged from 49.9% to 95.1% accuracy, with substantial variation driven primarily by training sample size and model architecture. LSTM-based models consistently achieved the highest accuracy, especially when trained with larger datasets. Pretrained embeddings and deeper LSTM architectures boosted performance in cases with moderate to large training sizes, while models trained on very small datasets produced weak or near-chance results.

2. Detailed Interpretation

2.1 Effect of Training Sample Size

Training data size was the most influential factor across all experiments.

- Very small datasets (100 samples) produced accuracies near chance levels (Models 1 and 8 at 51.3% and 59.6%, respectively). The models lacked sufficient examples to learn meaningful patterns.
- Moderate datasets (10,000–16,000 samples) showed marked improvement. For example:
  - Embedded Layer models increased from 50% → 89.3% when training size rose from 10,000 to 16,000 samples.
  - LSTM models showed strong improvements, reaching 82.98% at 16,000 samples.
- Large datasets (30,000–32,000 samples) yielded the highest performance:
  - Model 6 (LSTM) reached 95.1% accuracy with 32,000 samples.
  - Model 10 (Pretrained 2-layer LSTM) reached 88.1% with 30,000 samples.

Interpretation:
These results align with established deep learning theory: larger datasets significantly improve model generalization, while small datasets limit the model's ability to capture underlying text patterns.

2.2 Effect of Input Representation

Various input encoding strategies were explored:

- Embedded layer models improved significantly with more data, indicating they benefit from learned semantic representations.

- One-hot encoding (Model 4) performed surprisingly well (87.9%) with 10,000 samples. Although computationally expensive, one-hot representations can perform strongly on smaller vocabularies or well-structured datasets.
- Masking-enabled model (Model 7) produced near-chance performance (49.9%). This suggests either:
  - Masked tokens reduced essential input information, or
  - Masking was not aligned with the task requirements.

Interpretation:
Embedding layers generally provide more flexibility than one-hot encodings, but one-hot representations can still be competitive. Masking requires careful implementation and may degrade performance when improperly applied.

2.3 Effect of Model Architecture

There were clear performance differences based on architecture:

- Basic embedded models (Models 1–3) showed improvements largely driven by data size rather than model complexity.
- LSTM models (Models 5 and 6) demonstrated superior performance due to their ability to capture sequential dependencies. Model 6 achieved the highest accuracy of 95.1%, highlighting LSTM strength when provided sufficient data.
- Pretrained LSTM architectures (Models 9 and 10) showed additional gains, particularly with larger training sets:
  - 4-layer pretrained LSTM: 67.2% (15,000 samples)
  - 2-layer pretrained LSTM: 88.1% (30,000 samples)
- GloVe pretrained embeddings (Model 8) did not outperform custom embeddings, because training data was extremely limited (100 samples), preventing effective fine-tuning.

Interpretation:
Model complexity matters, but only when backed by adequate data. LSTMs excel at sequence modeling, and pretraining yields benefits primarily when the model receives enough data to adapt to the target task.

2.4 Test Loss Patterns

Loss values closely mirrored accuracy trends:

- High loss (~0.69) in low-performance models indicates the model is uncertain and outputting near-random predictions.
- Low loss (~0.18–0.47) in higher-accuracy models signals strong confidence and well-formed decision boundaries.

Interpretation:
The consistency between accuracy and loss validates the reliability of results across all models.

3. Conclusions

1. Training Sample Size Is the Primary Driver of Model Performance:
   Models with small training sets (<10,000 samples) consistently underperformed, often near chance levels. Significant performance improvements emerged only after training size exceeded 15,000 samples.
2. LSTM Architectures Provide Superior Accuracy:
   LSTMs outperformed simple embedding and one-hot models, particularly when trained with large datasets. Model 6 (LSTM with 32,000 samples) achieved the overall best performance with 95.1% accuracy.
3. Pretraining Offers Benefits but Requires Sufficient Data:
   Pretrained GloVe or LSTM models do not inherently guarantee high performance. Their advantages become evident only with moderate to large training sizes.
4. Masking Must Be Applied Carefully:
   Masking significantly reduced performance, suggesting that either valuable information was masked or the masking strategy was not aligned with model needs. This emphasizes the importance of proper preprocessing.
5. One-Hot Encoding Can Still Compete in Certain Settings:
   Despite being more primitive, the one-hot model performed strongly (87.9%), showing that simple representations can still be effective for specific tasks or datasets.
6. **Overall Reliability of Results:**
   All results are coherent, consistent with theoretical expectations, and indicate that the models behaved as expected given their architecture and training conditions.