# Sentiment Analysis using RNN Architectures and Attention Mechanisms

Pathri Vidya Praveen — Epoch, IIT Hyderabad

July 6, 2025

## 1 Introduction

This project explores sentiment classification using Recurrent Neural Networks (RNNs) and various attention mechanisms on a text dataset. We use a subset of the IMDB movie review dataset to train, evaluate, and compare 20 models, analyzing how attention improves performance and interpretability. The project also includes attention visualization and experiments with pretrained word embeddings (GloVe).

## 2 Objectives

- Build and compare different RNN-based architectures (Vanilla RNN, Vanilla LSTM, Bidirectional RNN, Bidirectional LSTM).
- Implement and integrate 4 attention mechanisms (Bahdanau(Additive), Luong Dot, Luong General, Luong Concat).
- Evaluate performance with and without attention on a sentiment classification task.
- Visualize attention weights to understand model focus.
- Use pretrained word embeddings (GloVe) to improve generalization.

## 3 Dataset

**Dataset used:** IMDB movie review dataset from HuggingFace `datasets` library.

- 25K training samples + 25K test samples merged into 50K total.
- Sampled 30K reviews for the project (due to GPU limits).
- Re-split into 24K training, 6K test.
- Labels: `0 = negative`, `1 = positive`.

# 4 Preprocessing and Embedding

- Tokenization done using basic whitespace splitting and stopwords removal.
- Vocabulary built from training data.
- Used **GloVe 6B 100d** pretrained word embeddings to initialize the embedding matrix.
- Unknown words are initialized with the mean of all GloVe vectors.

# 5 Model Architectures

## 5.1 Base RNN Models

Each model follows a common structure:

- Embedding layer (pretrained GloVe).
- Recurrent Layer (Vanilla RNN / Vanilla LSTM / Bidirectional RNN / Bidirectional LSTM).
- Pooling: Last hidden state or attention.
- Fully Connected (Linear + Sigmoid).

## 5.2 Attention Mechanisms

We integrated 4 custom attention mechanisms:

1) **Bahdanau (Additive Attention)**: Computes alignment score using learnable parameters and a feedforward network.
2) **Luong Dot**: Simple dot product between query and keys.
3) **Luong General**: Dot product after applying a learnable linear transformation.
4) **Luong Concat**: Concatenates hidden state and encoder outputs and applies a feedforward network.

## 5.3 Final Models

Total of 20 models:

- 4 without attention: Vanilla RNN, Vanilla LSTM, Bidirectional RNN, Bidirectional LSTM.
- 16 with attention: Each of the 4 RNN types + 4 attention types. (Simply cross product!)

# 6 Training Details

- Framework: PyTorch
- Optimizer: Adam
- Loss: Binary Cross Entropy
- Epochs: 5–10 (varies by model). 5 epochs used here.
- Batch size tuned based on GPU memory
- Logging and visualization enabled

# 7 Evaluation Metrics

- Accuracy
- Precision, Recall
- F1-Score (Micro and Macro)

# 8 Attention Mechanisms — Deep Dive

## 8.1 Bahdanau Attention (Additive)

- Introduced in the original seq2seq with attention paper.
- Alignment score is computed using:

$$score(h_t, s) = v^\top \tanh(W_1 h_t + W_2 s)$$

  where $h_t$ is the encoder output and $s$ is the decoder hidden state.

- A softmax is applied over scores to get weights. Weighted sum of encoder outputs forms the context.
- **Implemented using:** Linear layers for score computation + batch matrix operations.

## 8.2 Luong Dot Attention

- Alignment is simply:

$$score(h_t, s) = h_t^\top s$$

- Fastest among all due to no parameters.
- **Implemented using:** Batch matrix multiplication.

## 8.3 3. Luong General Attention

- Adds a learnable matrix $W$ to transform decoder hidden state:

$$score(h_t, s) = h_t^\top W s$$

- More flexible than dot but still efficient.
- **Implemented using:** Linear projection before dot product.

## 8.4 4. Luong Concat Attention

- Concatenates encoder and decoder hidden states:

$$score(h_t, s) = v^\top \tanh(W[h_t; s])$$

- Most expressive Luong variant; slightly slower due to concatenation and projection.
- **Implemented using:** Concatenation + Linear layer.

## 8.5 Summary of Findings and Learnings

- Attention mechanisms significantly improve model focus and learning capacity.
- Bahdanau and Concat were more powerful but slower.
- Dot attention is simple and efficient — best for speed-critical tasks.
- Visualizing attention weights gave interpretability to otherwise black-box models.
- Understanding and implementing attention manually (without libraries) deepened intuition on how NLP models really work under the hood.

Table 1: Accuracy Comparison Across Models

| **Model** | **None** | **Bahdanau** | **Luong Dot** | **Luong General** | **Luong Concat** |
|---|---|---|---|---|---|
| Vanilla RNN | 67.47% | 85.82% | 79.68% | 84.62% | 87.07% |
| Vanilla LSTM | 86.08% | 87.05% | 86.85% | 84.88% | 85.93% |
| Bidirectional RNN | 78.80% | 87.98% | 79.30% | 70.98% | 86.78% |
| Bidirectional LSTM | 86.42% | 87.25% | 87.35% | 86.22% | 87.40% |

- Maximum accuracy - Bidirectional RNN + Bahdanau Attention
- Maximum F1 score - Bidirectional RNN + Bahdanau Attention
- Maximum Precision - Bidirectional RNN + Bahdanau Attention
- Maximum Recall - Bidirectional RNN + Bahdanau

I

Table 2: F1 Score Comparison Across Models

| Model | None | Bahdanau | Luong Dot | Luong General | Luong Concat |
|---|---|---|---|---|---|
| Vanilla RNN | 66.16% | 85.75% | 79.60% | 84.62% | 87.06% |
| Vanilla LSTM | 86.08% | 87.04% | 86.85% | 84.77% | 85.86% |
| Bidirectional RNN | 78.80% | 87.98% | 79.02% | 70.79% | 86.76% |
| Bidirectional LSTM | 86.41% | 87.25% | 87.35% | 86.16% | 87.40% |

Table 3: Precision Comparison Across Models

| Model | None | Bahdanau | Luong Dot | Luong General | Luong Concat |
|---|---|---|---|---|---|
| Vanilla RNN | 70.65% | 86.45% | 80.19% | 84.62% | 87.14% |
| Vanilla LSTM | 86.14% | 87.15% | 86.85% | 85.96% | 86.69% |
| Bidirectional RNN | 78.81% | 88.03% | 80.95% | 71.56% | 87.09% |
| Bidirectional LSTM | 86.48% | 87.27% | 87.36% | 86.86% | 87.40% |

Table 4: Recall Comparison Across Models

| Model | None | Bahdanau | Luong Dot | Luong General | Luong Concat |
|---|---|---|---|---|---|
| Vanilla RNN | 67.47% | 85.82% | 79.68% | 84.62% | 87.07% |
| Vanilla LSTM | 86.08% | 87.05% | 86.85% | 84.88% | 85.93% |
| Bidirectional RNN | 78.80% | 87.98% | 79.30% | 70.98% | 86.78% |
| Bidirectional LSTM | 86.42% | 87.25% | 87.35% | 86.22% | 87.40% |

# 9 Results and Observations

- **Attention-based models consistently outperform non-attention baselines.**
- **Bidirectional RNN + Bahdanau** yielded the best performance surprisingly!! Though expected Bidirectional LSTM + Bahdanau to be the best model . This maybe because of some training variance , dropout and overfititing , shorter reviews maybe. We can tune the code's hyperparameters.
- Bahdanau and Concat are generally more expressive but slower.
- Attention visualizations show meaningful word-level focus.

# 10 Visualization

In the attention_outputs folder we have taken 3 samples each for all the 16 attention based models and shown the attention weights for each word . This is really beneficial to understand which words weight more in sentiment analysis classification and to understand importance of each individual word in the text.

# 11 Modular Code Structure

- `utils.py` — Tokenization, padding, vocab, and dataloaders.
- `models/base_models.py` — All the 4 base models along with attention wrapper
- `main.py` — Training, evaluation, saving.
- `train.py` — CLI and runner script.
- `visualize_attention_weights.py` — Individual attention weights visualization for 3 samples each in 16 models.
- `attention/bahdanau.py` - Implemented Bahdanau or Additive attention
- `attention/luong_dot.py` - Implemented Luong Dot Attention
- `attention/luong_general.py` - Implemented Luong General Attention
- `attention/luong_concat.py` - Implemented Luong Concat Attention

# 12 Improvements

- Instead of using basic whitespace splitting in the model , we can use some tokenizer like SpaCy or BertTokenizer
- I have used macro averaging here - Micro averaging is another strategy that can be used .
- Hyperparameter tuning can be done using optuna (Bayesian optimization) - to be explored later
- Train with Curriculum Learning i.e Start with shorter sequences and gradually increase to longer ones over epochs. This often helps RNNs learn better generalizations early.
- Attention Visualization with Heatmaps
- Layer Normalization / Dropout
- Self-Attention (Transformer-lite). Allows global context modeling.

  Improves performance even on small datasets.
- Switch to FastText or contextual embeddings.
- Custom Vocabulary Pruning.
- Transformer network based models can be highly useful.

# 13 Conclusion

This project provides an in-depth analysis of attention in RNNs for NLP. It highlights the power of attention mechanisms in improving model performance and interpretability. Future extensions could explore Transformer-based models, dynamic attention masking, or multilingual sentiment tasks.

# Credits

- IMDB Dataset: IMDB — HuggingFace Datasets
- GloVe embeddings: GloVe 6B (Stanford)