Sentiment Analysis with Attention Mechanisms: Project Report

July 1, 2025

Abstract

This report presents a sentiment analysis project on the IMDB dataset, implementing and comparing 20 neural network models, including Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), Bidirectional RNN (BiRNN), and Bidirectional LSTM (BiLSTM), with and without four attention mechanisms: Bahdanau, Luong Dot, Luong General, and Luong Concat. Each attention mechanism is described, and their PyTorch implementations are detailed. The report includes textual descriptions of model architectures, training and evaluation processes, and a summary of findings and key learnings from the experiments.

1 Introduction

This project conducts binary sentiment classification (positive or negative) on the IMDB dataset using PyTorch. Four base architecturesRNN, LSTM, BiRNN, and BiLSTMwere implemented, each with and without attention mechanisms to enhance focus on relevant input tokens. The dataset, consisting of 25,000 training and 25,000 test examples, was preprocessed to create a vocabulary, convert text to sequences, and pad sequences to a fixed length of 500 tokens. Twenty model variants were trained and evaluated, with performance visualized through loss curves, gradient norms, confusion matrices, and attention weight distributions.

2 Attention Mechanisms: Description and Implementation

Attention mechanisms enable models to assign weights to input tokens based on their relevance to the task. This section describes the four implemented attention mechanismsBahdanau, Luong Dot, Luong General, and Luong Concatand their PyTorch implementations.

2.1 Bahdanau Attention

- **Description**: Introduced by **(author?)** (1), Bahdanau attention is an additive mechanism that computes alignment scores by combining encoder outputs and a query (e.g., RNN hidden state) through a feedforward neural network. Scores are computed using a learnable weight matrix and vector, followed by a softmax operation to obtain attention weights.
- **Formulas**:

$$e_{t,i} = v_a^\top \tanh(W_a[h_i; s_t]) \quad (1)$$

$$\alpha_{t,i} = \text{softmax}(e_{t,i}) \quad (2)$$

$$c_t = \sum_i \alpha_{t,i} h_i \quad (3)$$

  where h_i is the encoder output, s_t is the query, W_a is a weight matrix, and v_a is a learnable vector.
- **Implementation**: Implemented as a PyTorch module (`BahdanauAttention`) with inputs `encoder_outputs` ($[batch\_size, seq\_len, hidden\_dim]$) and `query` ($[batch\_size, hidden\_dim]$).

A linear layer (`W_a`) combines concatenated encoder outputs and query, followed by a tanh activation and dot product with `v_a`. Outputs are a context vector ($[batch\_size, hidden\_dim]$) and attention weights ($[batch\_size, seq\_len]$).

- **Code**:

```python
class BahdanauAttention(nn.Module):
    def __init__(self, hidden_dim):
        super().__init__()
        self.hidden_dim = hidden_dim
        self.W_a = nn.Linear(2 * hidden_dim, hidden_dim)
        self.v_a = nn.Parameter(torch.randn(hidden_dim))
    def forward(self, encoder_outputs, query):
        batch_size, seq_len, enc_dim = encoder_outputs.size()
        query_expanded = query.unsqueeze(1).expand(batch_size, seq_len
            , enc_dim)
        combined = torch.cat((encoder_outputs, query_expanded), dim=2)
        scores = torch.tanh(self.W_a(combined))
        attention_scores = torch.matmul(scores, self.v_a)
        attention_weights = F.softmax(attention_scores, dim=1)
        context = torch.bmm(attention_weights.unsqueeze(1),
            encoder_outputs).squeeze(1)
        return context, attention_weights
```

2.2 Luong Dot Attention

- **Description**: Proposed by **(author?)** (2), this multiplicative attention computes alignment scores as the dot product between the query and encoder outputs. It is parameter-free and computationally efficient.
- **Formulas**:

$$e_{t,i} = s_t^\top h_i \tag{4}$$

$$\alpha_{t,i} = \text{softmax}(e_{t,i}) \tag{5}$$

$$c_t = \sum_i \alpha_{t,i} h_i \tag{6}$$

- **Implementation**: Implemented as `LuongDotAttention`, using batch matrix multiplication for the dot product, followed by softmax. No learnable parameters are required.
- **Code**:

```python
class LuongDotAttention(nn.Module):
    def forward(self, encoder_outputs, query):
        batch_size, seq_len, enc_dim = encoder_outputs.size()
        scores = torch.bmm(query.unsqueeze(1), encoder_outputs.
            transpose(1, 2)).squeeze(1)
        attention_weights = F.softmax(scores, dim=1)
        context = torch.bmm(attention_weights.unsqueeze(1),
            encoder_outputs).squeeze(1)
        return context, attention_weights
```

2.3 Luong General Attention

- **Description**: A variant of Luong attention that applies a linear transformation to encoder outputs before computing the dot product, introducing learnable parameters for flexibility.

- **Formulas**:

$$e_{t,i} = s_t^\top W_a h_i \tag{7}$$
$$\alpha_{t,i} = \text{softmax}(e_{t,i}) \tag{8}$$
$$c_t = \sum_i \alpha_{t,i} h_i \tag{9}$$

- **Implementation**: Implemented as `LuongGeneralAttention` with a linear layer (`W_a`) to transform encoder outputs.
- **Code**:

```
class LuongGeneralAttention(nn.Module):
    def __init__(self, hidden_dim):
        super().__init__()
        self.W_a = nn.Linear(hidden_dim, hidden_dim, bias=False)
    def forward(self, encoder_outputs, query):
        batch_size, seq_len, enc_dim = encoder_outputs.size()
        transformed = self.W_a(encoder_outputs)
        scores = torch.bmm(query.unsqueeze(1), transformed.transpose
            (1, 2)).squeeze(1)
        attention_weights = F.softmax(scores, dim=1)
        context = torch.bmm(attention_weights.unsqueeze(1),
           encoder_outputs).squeeze(1)
        return context, attention_weights
```

2.4 Luong Concat Attention

- **Description**: Another Luong variant that concatenates the query and encoder outputs, processes them through a linear layer, and computes scores using a learnable vector.
- **Formulas**:

$$e_{t,i} = v_a^\top \tanh(W_a[h_i; s_t]) \tag{10}$$
$$\alpha_{t,i} = \text{softmax}(e_{t,i}) \tag{11}$$
$$c_t = \sum_i \alpha_{t,i} h_i \tag{12}$$

- **Implementation**: Implemented as `LuongConcatAttention` with a linear layer (`W_a`) and learnable vector (`v_a`).
- **Code**:

```
class LuongConcatAttention(nn.Module):
    def __init__(self, hidden_dim):
        super().__init__()
        self.W_a = nn.Linear(2 * hidden_dim, hidden_dim)
        self.v_a = nn.Parameter(torch.randn(hidden_dim))
    def forward(self, encoder_outputs, query):
        batch_size, seq_len, enc_dim = encoder_outputs.size()
        query_expanded = query.unsqueeze(1).expand(batch_size, seq_len
            , enc_dim)
        combined = torch.cat((encoder_outputs, query_expanded), dim=2)
        scores = torch.tanh(self.W_a(combined))
        attention_scores = torch.matmul(scores, self.v_a)
        attention_weights = F.softmax(attention_scores, dim=1)
        context = torch.bmm(attention_weights.unsqueeze(1),
            encoder_outputs).squeeze(1)
        return context, attention_weights
```

3 Model Architectures

Twenty models were implemented: four base models (RNN, LSTM, BiRNN, BiLSTM) and their variants with each attention mechanism, resulting in 16 attention-based models.

3.1 Base Models

- **RNN**: Embedding layer ($[vocab\_size, embedding\_dim = 100]$), RNN layer ($[embedding\_dim, hidden\_dim = 128]$), and linear layer ($[hidden\_dim, 1]$). Outputs a tensor of shape $[batch\_size, 1]$.
- **LSTM**: Similar to RNN but uses an LSTM layer for long-term dependencies.
- **BiRNN**: Bidirectional RNN with doubled hidden dimension ($2 \times hidden\_dim$). Concatenates the last forward and first backward hidden states.
- **BiLSTM**: Bidirectional LSTM, similar to BiRNN.

3.2 Attention Models

- **RNN/LSTM with Attention**: Adds an attention layer after the RNN/LSTM, using the last hidden state as the query. Outputs a tuple ($[batch\_size, 1]$, $[batch\_size, seq\_len]$).
- **BiRNN/BiLSTM with Attention**: Uses bidirectional outputs and a concatenated query (forward and backward hidden states). Attention operates on doubled hidden dimensions.

3.3 Architecture Diagrams

Diagrams are described below, as external images are not included. To visualize, use a tool like Draw.io.

- **RNN**: Input $\rightarrow$ Embedding $\rightarrow$ RNN $\rightarrow$ Linear $\rightarrow$ Sigmoid Output.
- **RNN with Attention**: Input $\rightarrow$ Embedding $\rightarrow$ RNN $\rightarrow$ Attention (query: last hidden state) $\rightarrow$ Linear $\rightarrow$ Sigmoid Output. Attention weights visualized as a heatmap.
- **BiRNN/BiLSTM with Attention**: Input $\rightarrow$ Embedding $\rightarrow$ BiRNN/BiLSTM $\rightarrow$ Concat (forward/backward) $\rightarrow$ Attention $\rightarrow$ Linear $\rightarrow$ Sigmoid Output.

4 Implementation Details

- **Dataset**: IMDB dataset (25,000 training, 25,000 test examples) from the `datasets` library.
- **Preprocessing**: Tokenized text using word splitting, built a vocabulary with `<PAD>` and `<UNK>`, padded sequences to 500 tokens, and created `DataLoader` with batch size 32.
- **Training**: Used `BCEWithLogitsLoss` and Adam optimizer (`lr=0.001`, 5 epochs). Handled tuple outputs for attention models and tensor outputs for base models.
- **Evaluation**: Computed accuracy, precision, recall, and F1 scores (macro and micro). Visualized loss, gradient norms, confusion matrices, and attention weights.

5 Summary of Findings and Key Learnings

5.1 Findings

- **Performance**: Attention models outperformed base models, with BiLSTM variants (e.g., `BiLSTM_Bahdanau`, `BiLSTM_General`) achieving the highest accuracy and F1 scores due to bidirectional context and long-term dependency modeling.
- **Attention Weights**: Visualizations showed higher weights on sentiment-related words (e.g., “great”, “terrible”). Bahdanau and Luong Concat produced diverse weight distributions, while Luong Dot was more uniform.
- **Training Stability**: LSTM-based models showed stable gradients, while RNN models exhibited occasional spikes, indicating potential vanishing/exploding gradient issues.

5.2 Key Learnings

- Attention mechanisms improve performance by focusing on relevant tokens, with Bahdanau and Luong Concat being more expressive but computationally intensive.
- BiLSTM with attention is well-suited for sentiment analysis due to its ability to capture bidirectional context.
- Robust output handling (tuple vs. tensor) was critical to avoid runtime errors, resolved by checking model name suffixes.
- Visualizations provided valuable insights into model behavior and training dynamics.
- Future improvements include using pre-trained embeddings (e.g., GloVe), adding dropout, or reducing vocabulary size for efficiency.

6 Conclusion

This project successfully implemented and compared 20 models for sentiment analysis, demonstrating the effectiveness of attention mechanisms. BiLSTM with Bahdanau or General attention achieved the best performance, highlighting the importance of bidirectional context and learned attention weights. Robust implementation and visualizations were key to understanding and improving the models. Future work could explore pre-trained embeddings or Transformer-based architectures.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *arXiv preprint arXiv:1508.04025*, 2015.