**Assignment: Long Short-Term Memory (LSTM)**

**Submitted By: Rajesh Kumar Sharma**

**Date: 27-Sep-24**

**Principles of Recurrent Neural Network (RNN)** :

**Recurrent Neural Networks (RNNs)** are a type of neural network designed to handle **sequential data**. Unlike traditional feedforward neural networks, RNNs have connections that loop back on themselves, allowing them to process and retain information from previous time steps. Here's an overview of their key principles:

1. **Sequential Data Processing**:
   * RNNs are built to process sequences of data where the order of input matters, such as time-series data, text, or audio. They apply the same set of weights recursively over each time step in a sequence, which allows them to "remember" previous inputs while processing the current one.
2. **Hidden State**:
   * RNNs maintain a **hidden state** that acts as a memory of previous inputs in the sequence. At each time step, the network updates its hidden state using the current input and the hidden state from the previous time step, allowing it to maintain context over the sequence.
3. **Shared Parameters**:
   * RNNs share the same weights across different time steps, meaning the same parameters are applied to each part of the input sequence. This is important for handling variable-length sequences and reducing the number of parameters needed for training.
4. **Backpropagation Through Time (BPTT)**:
   * RNNs are trained using **backpropagation through time**, which is a modified version of backpropagation for sequential data. In BPTT, gradients are propagated back through the entire sequence, allowing the network to adjust weights based on long-term dependencies.
5. **Short-Term Memory**:
   * Standard RNNs tend to have a **short memory** and struggle to retain information over long sequences due to the **vanishing gradient problem**. This occurs when gradients used to update the model during training become extremely small, making it difficult for the network to learn long-term dependencies.

**Principles of Long Short-Term Memory (LSTM)** :

***Long Short-Term Memory (LSTM)*** networks are a special type of RNN designed to overcome the limitations of standard RNNs, particularly the vanishing gradient problem. LSTMs can capture both **short-term and long-term dependencies** in sequential data. Here are their key principles:

1. **Cell State (Memory Cell)**:
   * LSTMs introduce a **cell state**, which runs through the entire sequence with minimal changes. This acts as a long-term memory, storing important information over long time periods. The cell state helps mitigate the vanishing gradient problem by allowing information to flow unimpeded unless explicitly modified by gates.
2. **Gating Mechanism**:
   * LSTMs use three types of gates (input, forget, and output) to control the flow of information, allowing the network to selectively remember or forget data.
   * **Forget Gate**: Decides which information should be discarded from the cell state. It takes the current input and the previous hidden state and outputs a value between 0 and 1 for each number in the cell state, where 0 means "completely forget" and 1 means "completely keep."
   * **Input Gate**: Controls how much of the new input should be added to the cell state. It helps update the cell state with new information.
   * **Output Gate**: Determines how much of the current cell state should be passed to the next hidden state and used as output.
3. **Selective Memory Retention**:
   * The gating mechanism allows LSTMs to selectively retain or discard information, which is crucial for capturing long-term dependencies in sequential data. This flexibility helps the network decide what to keep in memory and what to forget, depending on the task at hand.
4. **Handling Long-Term Dependencies**:
   * By overcoming the vanishing gradient problem through controlled gating, LSTMs are highly effective in learning and capturing **long-term dependencies** in sequences. This makes them suitable for tasks where context from earlier time steps is crucial for future predictions.
5. **Maintaining a Balance Between Memory and Output**:
   * The **hidden state** in LSTMs provides short-term memory, while the **cell state** holds long-term memory. The combination of these two allows LSTMs to maintain both recent and long-range information effectively, giving them flexibility in sequence modeling.
6. **Bidirectional LSTM (BiLSTM)**:
   * LSTMs can be extended to **bidirectional LSTMs**, where the network processes the input sequence both forwards and backwards, capturing dependencies in both directions. This is especially useful in tasks like machine translation and speech recognition, where future context can also be important.

**Advantages of LSTM in capturing long-term dependencies in sequential data:**

1. **Addressing the Vanishing Gradient Problem:**

* LSTMs mitigate Vanishing Gradient Problem by using a gating mechanism (input gate, forget gate, and output gate), which regulates the flow of information. This allows the network to keep gradients stable over long time steps, enabling it to capture long-term dependencies more effectively.

**2. Memory Cells for Long-Term Retention:**

* LSTMs have memory cells that maintain information over long periods. Unlike standard RNNs, LSTMs can remember relevant information for hundreds or thousands of time steps, which is crucial in tasks like language modeling, speech recognition, or time series prediction.
* This ability to preserve relevant information and discard irrelevant data over time helps LSTMs model sequential data more effectively.

**3. Selective Information Retention and Forgetting:**

* The forget gate in LSTMs decides what information to retain or forget, which is a significant improvement over standard RNNs. This selective forgetting enables LSTMs to remove irrelevant information from the memory cell, thus improving the model's efficiency and focus on important long-term dependencies.

**4. Handling Variable-Length Sequences:**

* LSTMs are well-suited for handling variable-length sequences, which is common in tasks like text, video, or time series analysis. Their gating mechanism allows them to manage dependencies across both short and long sequences effectively, which is harder for traditional RNNs.

**5. Better Context Understanding:**

* By using multiple LSTM layers (stacked LSTMs), the model can capture more complex and nuanced long-term dependencies. Higher layers can focus on broader, more abstract patterns in the data, while lower layers capture more local dependencies.

**6. Effective in Real-World Applications:**

* Due to their ability to handle long-term dependencies, LSTMs have been successfully applied in various fields:
  + Natural Language Processing (NLP): for tasks like machine translation, text generation, and sentiment analysis.
  + Time Series Forecasting: in predicting stock prices, weather patterns, or economic trends.
  + Speech Recognition: capturing long sequences of phonemes or words.
  + Video Processing: understanding context in sequences of frames.

**7. Bidirectional LSTMs:**

* Bidirectional LSTMs process the sequence in both forward and backward directions, capturing dependencies in both past and future contexts. This ability further enhances the model's understanding of long-term dependencies.

**8. Robustness to Noisy Data:**

* LSTMs can ignore irrelevant or noisy parts of a sequence, which improves their ability to generalize better on real-world data with inherent noise or inconsistency.

**Differences between RNN and LSTM**:

|  |  |  |
| --- | --- | --- |
| **Category** | **RNN** | **LSTM** |
| Memory | Short-term memory; struggles with long-term dependencies. | Can capture both short-term and long-term dependencies. |
| Vanishing Gradient Problem | Affected by vanishing gradient, limiting the ability to learn long-range patterns. | Uses gates to prevent vanishing gradient issues, enabling long-range learning. |
| Architecture | Simple architecture with hidden states. | Complex architecture with memory cells and gates. |
| Performance | Works well for short sequences. | Works well for long and complex sequences. |
| Training | Uses Backpropagation Through Time (BPTT). | Uses a modified version of BPTT with gating mechanisms. |

**Code Link:** <https://github.com/rajeshksharmasls/GenAI/blob/master/GenAI/LSTM/lstm_assgn.ipynb>

**Dataset**: - LSTM Next Word Prediction Dataset

[https://www.kaggle.com/datasets/hakim11/lstm-next-word-prediction-data/data]

**Process:**

**Libraries and Preprocessing:**

**Libraries:** Various TensorFlow/Keras modules for building LSTM models, preprocessing text, and visualizing results.

**Data Loading and Cleaning:** The input text is loaded from a file, cleaned (removal of newlines, special characters, etc.), and tokenized.

**Sequence Generation:** Input sequences are generated by tokenizing sentences and progressively increasing token length for LSTM training. These sequences are padded to a uniform length for consistency.

**Model Building:**

*Baseline Model:*

3 stacked LSTM layers with 128 units.

An embedding layer to convert words to dense vectors.

A softmax output layer for multi-class classification (predicting the next word).

Model is compiled using categorical cross-entropy loss and Adam optimizer.

*Optimized Model:*

Similar structure with increased units in LSTM layers (256).

Dropout layers to prevent overfitting.

Dense intermediate layer before the final softmax output layer.

Uses the Adam optimizer with a learning rate of 0.001 for finer tuning.

**Training and Evaluation:**

**Training:** Both models are trained on the input data, with the baseline model trained using a batch size of 64 and the optimized model using 128. Accuracy and loss are tracked over 15 epochs.

**Evaluation:** After training, both models are evaluated on the same dataset to compare performance.

**Next-Word Prediction:**

A function is provided to predict the next word in a given text input. It generates words sequentially based on the trained model.

**Plotting:**

Accuracy and loss graphs are plotted for both models to visualize performance over epochs.

**Results:**

The baseline and optimized models are compared in terms of accuracy and loss.

**Snapshots of Baseline Model:**

**Model: "sequential"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩

│ embedding (Embedding) │ (None, 151, 100) │ 1,426,800 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ lstm (LSTM) │ (None, 151, 128) │ 117,248 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ lstm\_1 (LSTM) │ (None, 151, 128) │ 131,584 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ lstm\_2 (LSTM) │ (None, 128) │ 131,584 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ batch\_normalization │ (None, 128) │ 512 │

│ (BatchNormalization) │ │ │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

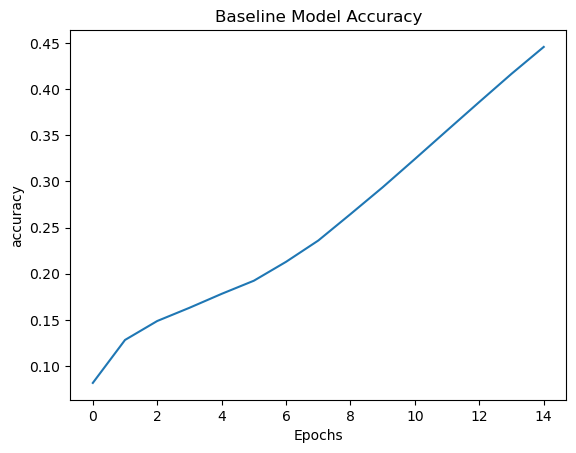
│ dense (Dense) │ (None, 7173) │ 925,317 │

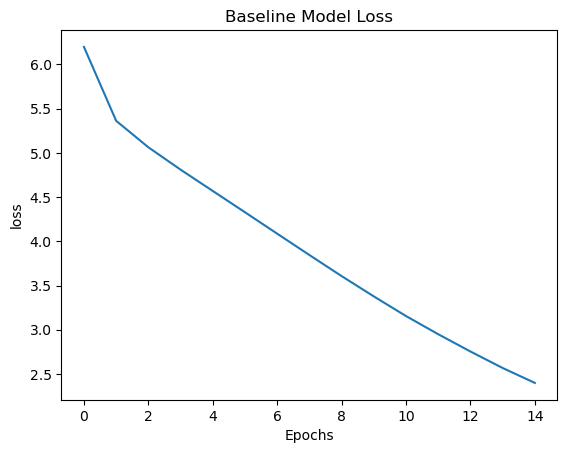
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘

**Total params:** 2,733,045 (10.43 MB)

**Trainable params:** 2,732,789 (10.42 MB)

**Non-trainable params:** 256 (1.00 KB)





**Snapshots of Optimized Model:**

**Model: "sequential\_1"**

**┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓**

**┃ Layer (type) ┃ Output Shape ┃ Param # ┃**

**┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩**

**│ embedding\_1 (Embedding) │ (None, 151, 128) │ 1,826,304 │**

**├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤**

**│ lstm\_3 (LSTM) │ (None, 151, 256) │ 394,240 │**

**├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤**

**│ dropout (Dropout) │ (None, 151, 256) │ 0 │**

**├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤**

**│ lstm\_4 (LSTM) │ (None, 151, 256) │ 525,312 │**

**├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤**

**│ dropout\_1 (Dropout) │ (None, 151, 256) │ 0 │**

**├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤**

**│ lstm\_5 (LSTM) │ (None, 256) │ 525,312 │**

**├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤**

**│ dense\_1 (Dense) │ (None, 1000) │ 257,000 │**

**├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤**

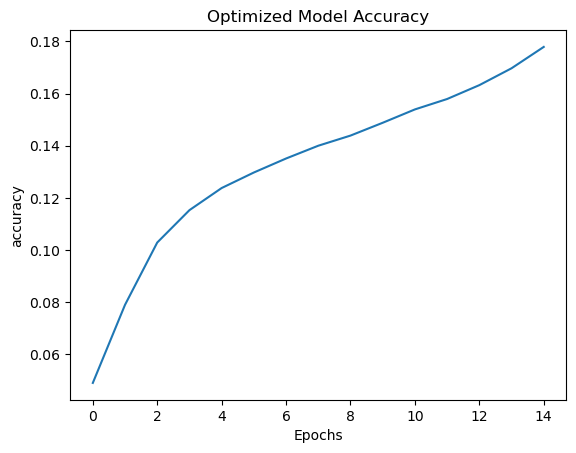
**│ dense\_2 (Dense) │ (None, 7173) │ 7,180,173 │**

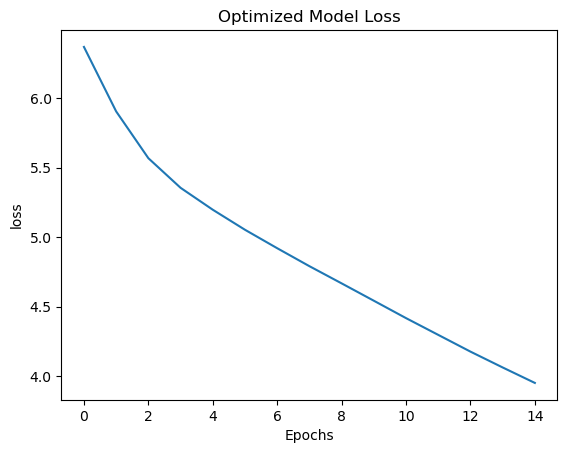
**└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘**

**Total params: 10,708,341 (40.85 MB)**

**Trainable params: 10,708,341 (40.85 MB)**

**Non-trainable params: 0 (0.00 B)**





**Next-Word Prediction System:**

**Baseline Model:** *who are the objects* of both self spreading between any other elegant goodness and she was now rendered sure

**Optimized Model:** *who are the objects* of the world and seek of the world and seek compassion to the other time

**Conclusion:**  Baseline Model has Accuracy of 55% and Loss of 1.89 whereas Optimized Model has Accuracy of 21% and Loss of 3.68. Hence, its clearly evident that the ***baseline model*** is already better than the optimized model in terms of accuracy and loss.