**Assignment: Recurrent Neural Network (RNN)**

**Submitted By: Rajesh Kumar Sharma**

**Date: 27-Sep-24**

**Recurrent Neural Network (RNN)**

**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.

**Different RNN architectures:**

**Vanilla :** Vanilla RNN (Recurrent Neural Network) is a type of neural network that is used for processing sequential data. It is the simplest type of RNN, where the hidden state at the current time step is determined by the input at the current time step and the hidden state from the previous time step.

One of the main issues with vanilla RNN is the vanishing gradient problem, where gradients propagated back through the network become extremely small, making it difficult to learn long-term dependencies in the input sequence. As a result, more advanced RNN architectures like LSTM and GRU were developed to address this issue.

**Gradient Recurrent Unit (GRU):** Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) that was introduced by Cho et al. in 2014 as a simpler alternative to Long Short-Term Memory (LSTM) networks. Like LSTM, GRU can process sequential data such as text, speech, and time-series data.

The basic idea behind GRU is to use gating mechanisms to selectively update the hidden state of the network at each time step. The gating mechanisms are used to control the flow of information in and out of the network. The GRU has two gating mechanisms, called the reset gate and the update gate.

The reset gate determines how much of the previous hidden state should be forgotten, while the update gate determines how much of the new input should be used to update the hidden state. The output of the GRU is calculated based on the updated hidden state.

**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.

**The advantages of Recurrent Neural Networks (RNNs) are:**

**Ability to Process Sequential Data:** RNNs can process sequential data of varying lengths, making them useful in applications such as natural language processing, speech recognition, and time-series analysis.

**Memory:** RNNs have the ability to retain information about the previous inputs in the sequence through the use of hidden states. This enables RNNs to perform tasks such as predicting the next word in a sentence or forecasting stock prices.

**Versatility:** RNNs can be used for a wide variety of tasks, including classification, regression, and sequence-to-sequence mapping.

**Flexibility:** RNNs can be combined with other neural network architectures, such as Convolutional Neural Networks (CNNs) or feedforward neural networks, to create hybrid models for specific tasks.

**However, there are also some disadvantages of RNNs:**

**Vanishing Gradient Problem:** The vanishing gradient problem can occur in RNNs, particularly in those with many layers or long sequences, making it difficult to learn long-term dependencies.

**Computationally Expensive:** RNNs can be computationally expensive, particularly when processing long sequences or using complex architectures.

**Lack of Interpretability:** RNNs can be difficult to interpret, particularly in terms of understanding how the network is making predictions or decisions.

Overall, while RNNs have some disadvantages, their ability to process sequential data and retain memory of previous inputs make them a powerful tool for many machine learning applications.

**Dataset:** Twitter Sentiment Analysis Dataset

[<https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis/data>]

**Code Link:** <https://colab.research.google.com/drive/1u8rXwkQ1e9coFcSsL4N1PPwti37E4TWq#scrollTo=F_nGurXf75jX>

<https://github.com/rajeshksharmasls/GenAI/blob/master/GenAI/RNN/rnn_assgn.py>

**Process:**

**Libraries Imported:** The code uses several libraries, including Pandas for data handling, Matplotlib and Seaborn for visualization, and machine learning libraries like Scikit-learn and Keras for model building and evaluation.

**Data Preprocessing:**

* Loads the Twitter sentiment dataset.
* Performs basic preprocessing such as removing duplicates and handling missing data.
* Conducts data visualization to explore the distribution of sentiments and the frequency of entities.

**Data Visualization:**

* Visualizes the distribution of sentiments (positive, negative, neutral, irrelevant) using bar graphs and pie charts.
* Generates word clouds for each sentiment category.
* Analyzes the length of messages and shows the frequency of entities.

**TF-IDF Vectorization:**

* Uses the TF-IDF method to convert text data into numerical form for machine learning models.

**K-Means Clustering:**

* Applies K-Means++ clustering to group similar messages.
* Visualizes clusters using Principal Component Analysis (PCA).

**Machine Learning Models:**

* Implements several machine learning algorithms (Naive Bayes, Logistic Regression, Random Forest, Decision Tree).
* Evaluates their performance using accuracy scores and classification reports.
* The Random Forest model achieves the highest accuracy (93.8%).

**Recurrent Neural Network (RNN):**

* Uses SimpleRNN layers to build a deep learning model for sentiment classification.
* Prepares the data in a 3D shape for RNN input and encodes target labels using one-hot encoding.
* The initial RNN model achieves a high accuracy of 96.7%.

**Optimization of RNN:**

* Tries to optimize the RNN model by adjusting its structure (e.g., adding dropout layers to prevent overfitting).
* However, the optimized RNN model achieves a lower accuracy (94%) compared to the initial RNN model.

**Accuracy & F1 Score for Vanilla and Optimized Models:**

RNN Model Accuracy: 0.9679679679679679

RNN Model Classification Report:

precision recall f1-score support

Irrelevant 0.97 0.95 0.96 171

Negative 0.98 0.97 0.98 266

Neutral 0.96 0.98 0.97 285

Positive 0.96 0.97 0.96 277

accuracy **0.97** 999

macro avg 0.97 0.97 0.97 999

weighted avg 0.97 0.97 0.97 999

Optimized RNN Model Accuracy: 0.9379379379379379

Optimized RNN Model Classification Report:

precision recall f1-score support

Irrelevant 0.95 0.91 0.93 171

Negative 0.92 0.96 0.94 266

Neutral 0.95 0.93 0.94 285

Positive 0.94 0.94 0.94 277

accuracy **0.94** 999

macro avg 0.94 0.94 0.94 999

weighted avg 0.94 0.94 0.94 999

**Conclusion:** Optimized RNN accuracy is 94% which is less than the Vanilla RNN accuracy of 97%. Hence, its implied that the Vanilla RNN itself is already optimized.