ASSIGNMENT No: 06

Title:- Perform sentiment analysis with a recurrent neural networks RNN

Mapping with Syllabus –

Unit 4

Objective –

Implement a Recurrent Neural Network (RNN) on a network graph for sentiment analysis.

Outcome –

Solve the language translation problem by Recurrent neural network(RNN)

Software Requirements –

- Python (3.x recommended)
- Jupyter Notebook or any Python IDE

Hardware Requirements –

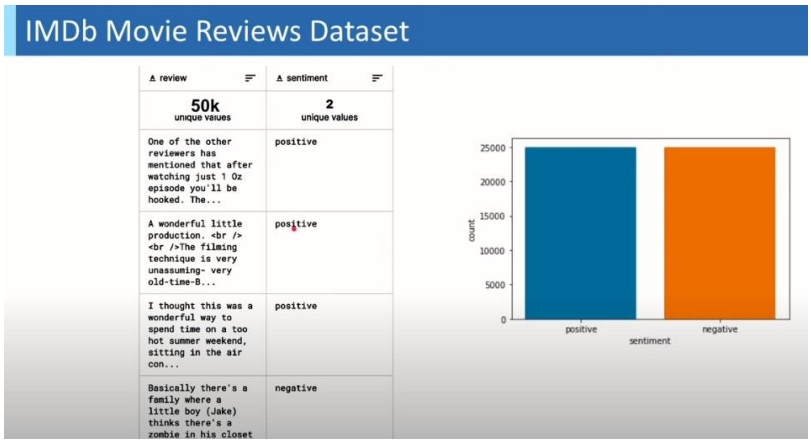
A machine with sufficient RAM and processing power for model training (8GB RAM recommended)

Prerequisites –

- Basic understanding of Python programming
- Familiarity with the concepts of Neural Networks

Dataset –

https://github.com/skillcate/sentiment-analysis-with-deep-neural-networks/blob/main/a1_IMDB_Dataset.csv

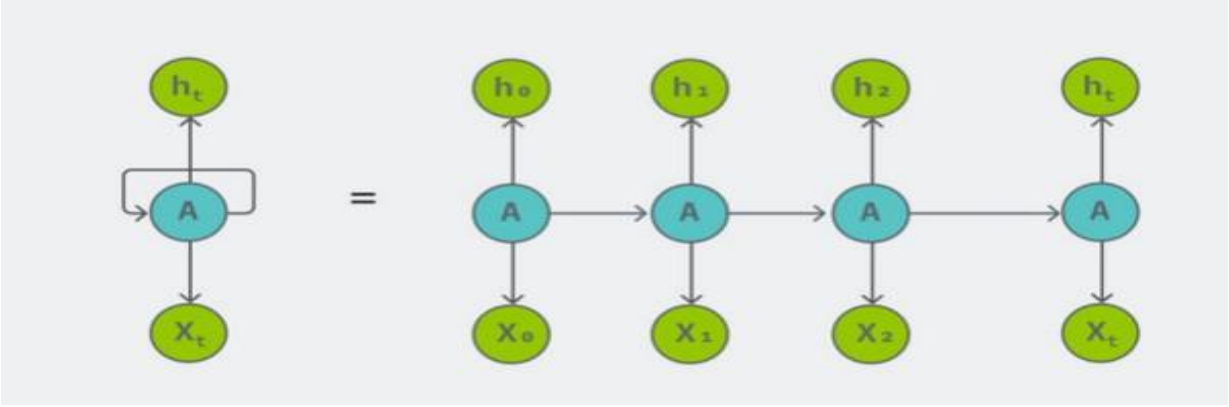


Libraries or Modules Used–

- TensorFlow or PyTorch
- NumPy
- Matplotlib

Theory –

A Recurrent Neural Network (RNN) represents a sophisticated extension of the traditional feedforward neural network by incorporating an internal memory mechanism. In contrast to feedforward networks, RNNs possess a recurrent nature, executing the same operation for each input data while considering the outcome of the preceding computation. This iterative process involves generating an output, which is then replicated and fed back into the recurrent network. In making decisions, the RNN takes into account both the current input and the knowledge acquired from the previous input.



RNNs, uniquely suited for tasks like unsegmented connected handwriting recognition or speech

recognition, leverage their internal state (memory) to effectively process sequences of inputs. Unlike other neural networks where inputs are treated independently, RNNs establish a relationship among all inputs in a sequence.

The sequence begins with the initial input, X(0), producing an output, h(0), which, together with X(1), becomes the input for the subsequent step. This process continues iteratively, with each hidden state (h) from the previous step being combined with the current input (X) for the next computation. This sequential progression ensures that the network retains contextual information throughout the training process.

The formula for the current state involves weight matrices (W) for the current input (Whx) and the previous hidden state (Whh), alongside an activation function, typically tanh. This activation function introduces non-linearity, compressing activations within the range of [-1, 1]. The final output state (Yt) is determined by the weight matrix (Why) at the output layer.

$$h_t = f(h_{t-1}, x_t)$$

Applying Activation Function:

$$h_t = \tanh (W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Yt is the output state. Why is the weight at the output state.

In essence, RNNs excel at learning and remembering dependencies between inputs across sequential data, making them invaluable for tasks that involve context and temporal relationships.

Algorithm –

Applying Recurrent Neural Networks (RNNs) to network graphs for sentiment analysis involves a step-by-step process. Here's a high-level algorithm:

1. Data Preparation:
   - Collect a dataset with network graph data and sentiment labels for each node or edge.
   - Represent the graph as an adjacency matrix or an edge list.
   - Assign sentiment labels to nodes or edges in the graph.

2. Node/Edge Embedding:
   - Represent each node or edge in the graph using embeddings. You can use techniques like Word2Vec, GloVe, or Graph Embeddings (e.g., GraphSAGE) to convert nodes into fixed-size vectors.

3. Sequence Generation:
   - Create sequences of node or edge embeddings for each path or subgraph in the network. The sequence length can vary based on the application and context.

4. Labeling:
   - Assign sentiment labels to the sequences based on the sentiment of the nodes or edges in the sequence. This is your target variable for training the RNN.

5. RNN Model Architecture:
   - Choose an RNN architecture suitable for sequence processing. Long Short-

rm Memory (LSTM) or Gated Recurrent Unit (GRU) cells are often used due to their ability to capture long-range dependencies.
   - Define the input layer to accept sequences of node or edge embeddings.
   - Optionally, include additional layers like Bidirectional RNNs or attention mechanisms for improved performance.

6. Training:
   - Split the dataset into training, validation, and test sets.
   - Train the RNN model on the training set using backpropagation through time (BPTT) or other sequence training methods.
   - Optimize the model using an appropriate loss function (e.g., categorical cross-entropy) and an optimizer (e.g., Adam).

7. Evaluation:
   - Evaluate the trained model on the validation set to tune hyperparameters and prevent overfitting.
   - Test the model on the test set to assess its performance on unseen data.

8. Inference:
   - Use the trained RNN model for sentiment analysis on new or unseen network graphs.

9. Post-Processing:
   - Analyze the results, and if needed, perform post-processing or fine-tuning to improve the model's accuracy.

10. Visualization (Optional):
   - Visualize the sentiment predictions on the network graph to gain insights into sentiment patterns and relationships.

Remember that the specifics of the algorithm may vary based on the exact nature of your network graph data and the sentiment analysis task at hand. Adjustments may be needed based on the characteristics of your dataset and the desired outcomes.

Let's understand using an example:
Let's break down the algorithm with a simplified example of sentiment analysis on a network graph. In this example, we'll consider a social network where nodes represent users, and edges represent connections between users. The task is to predict the sentiment (positive or negative) of interactions between users.

Example:

1. Data Preparation:
   - Collect a dataset with a social network graph, where each node represents a user and each edge represents a connection.
   - Assign sentiment labels (positive or negative) to edges based on the sentiment of interactions between users.

2. Node Embedding:
   - Represent each user (node) with Word2Vec embeddings based on their interactions or posts in the social network.

3. Sequence Generation:
   - Create sequences of node embeddings for each path of interactions between users. For example, if we have the following interactions: A -> B (positive), B -> C (negative), C -> D (positive), the sequence for user A could be [A_embedding, B_embedding].

4. Labeling:
   - Assign sentiment labels to the sequences. For instance, if the sequence is [A_embedding, B_embedding] and the sentiment of the interaction B -> C is negative, then the label for this sequence is negative.

5. RNN Model Architecture:
   - Use an LSTM-based RNN architecture for sequence processing.
   - Define the input layer to accept sequences of user embeddings.
   - Add LSTM layers to capture the temporal dependencies in the sequence.

6. Training:
   - Split the dataset into training, validation, and test sets.
   - Train the RNN model on the training set using backpropagation through time (BPTT).
   - Optimize the model using categorical cross-entropy as the loss function and the Adam optimizer.

7. Evaluation:
   - Evaluate the trained model on the validation set to tune hyperparameters and prevent overfitting.
   - Test the model on the test set to assess its performance on unseen interactions.

8. Inference:

- Use the trained RNN model for sentiment analysis on new interactions between users.

9. Post-Processing:
   - Analyze the results, and if needed, perform post-processing or fine-tuning to improve the model's accuracy.

10. Visualization (Optional):
    - Visualize the predicted sentiments on the social network graph to understand sentiment patterns and relationships between users.

This example outlines a simplified algorithm for sentiment analysis on a social network graph using an RNN. Keep in mind that real-world scenarios may involve more complex data preprocessing, model tuning, and feature engineering based on the characteristics of your dataset.

Application –

1. Social Media Sentiment Analysis:
   - ❖ Analyzing sentiments in interactions between users on social media platforms.
   - ❖ Identifying positive or negative sentiments in comments, replies, and discussions.

2. Online Community Monitoring:
   - ❖ Monitoring sentiments within online communities or forums.
   - ❖ Identifying potential issues, conflicts, or positive interactions within community discussions.

3. Brand Reputation Management:
   - ❖ Analyzing sentiments related to a brand by examining interactions between users mentioning the brand.
   - ❖ Monitoring sentiment trends over time and identifying areas for improvement.

4. Customer Feedback Analysis:
   - ❖ Analyzing sentiments in customer reviews and feedback.
   - ❖ Understanding the overall sentiment regarding a product or service.

5. Collaborative Filtering in Recommendation Systems:
   - ❖ Utilizing sentiments in user-item interactions to improve collaborative filtering recommendation systems.
   - ❖ Personalizing recommendations based on both historical interactions and sentiments.

6. Financial News Analysis:
   - ❖ Analyzing sentiments in financial news articles and discussions between investors.
   - ❖ Understanding market sentiment and potential impacts on stock prices.

7. Healthcare Interactions:
   - ❖ Analyzing sentiments in interactions between patients and healthcare professionals.
   - ❖ Identifying trends in patient satisfaction or areas for improvement in healthcare services.

8. Educational Platforms:
   - ❖ Analyzing sentiments in interactions between students and instructors on educational platforms.
   - ❖ Monitoring student engagement and identifying potential issues or areas for improvement.

9. Political Discourse Analysis:
   - ❖ Analyzing sentiments in interactions between individuals discussing political topics.
   - ❖ Identifying trends in public opinion and potential areas of concern.

10. Influencer Marketing:
    - ❖ Analyzing sentiments in interactions between influencers and their followers.
    - ❖ Understanding the impact of influencer content on audience sentiment.

These applications highlight the versatility of sentiment analysis using RNNs on network graphs, where the sequential nature of interactions and relationships plays a crucial role in understanding sentiments in various domains.

Inference –

For sentiment analysis with RNNs, new data is processed by a pre-trained model, generating predictions. Post-processing and analysis follow, with optional visualizations. A feedback loop may improve the model, allowing real-time sentiment analysis in production.

Code –

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb from
tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Set the parameters
max_features = 10000  # Number of words to consider as features
maxlen = 100  # Cut texts after this number of words (among top max_features most common words)
batch_size = 32

# Load the IMDB dataset
```

```
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# Pad sequences to have a consistent length for the input to the RNN x_train =
pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

# Build the RNN model with LSTM model
= Sequential()
model.add(Embedding(max_features, 128)) model.add(LSTM(64, dropout=0.2,
recurrent_dropout=0.2)) model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['accuracy'])

# Train the model model.fit(x_train,
y_train,
batch_size=batch_size, epochs=5,
validation_data=(x_test, y_test))



# Evaluate the model
score, acc = model.evaluate(x_test, y_test, batch_size=batch_size) print(f'Test score: {score}')
print(f'Test accuracy: {acc}')

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras- datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the criteria. It will use a
generic GPU kernel as fallback when running on GPU.
Epoch 1/5
782/782 [==============================] - 319s 401ms/step - loss: 0.4161 - accuracy:
0.8074 - val_loss: 0.3585 - val_accuracy: 0.8412
Epoch 2/5
782/782 [==============================] - 288s 368ms/step - loss: 0.2625 - accuracy:
0.8950 - val_loss: 0.3482 - val_accuracy: 0.8454
Epoch 3/5
782/782 [==============================] - 284s 363ms/step - loss: 0.1931 - accuracy:
0.9244 - val_loss: 0.4158 - val_accuracy: 0.8375
Epoch 4/5
782/782 [==============================] - 285s 365ms/step - loss: 0.1431 - accuracy:
0.9472 - val_loss: 0.4504 - val_accuracy: 0.8412
Epoch 5/5
782/782 [==============================] - 287s 367ms/step - loss: 0.1093 - accuracy:
0.9606 - val_loss: 0.4790 - val_accuracy: 0.8413
782/782 [==============================] - 25s 32ms/step - loss: 0.4790 - accuracy:
0.8413
Test score: 0.47897636890411377
Test accuracy: 0.8413199782371521
```

References –

https://www.geeksforgeeks.org/what-is-sentiment-analysis/amp/

https://www.google.com/amp/s/www.geeksforgeeks.org/introduction-to- recurrent-neural-network/amp/

https://www.analyticsvidhya.com/blog/2022/01/sentiment-analysis-with-lstm/

Conclusion: Thus Performed Sentiment Analysis in the network graph using RNN.