****Department of Computer Science and Engineering (CSE)

**Trimester : SPRING** 2023

**Course : CSE 4889 - Machine Learning**

**Section: A**

Assignment - 02

**Topic : How to train RNN**

**Name : Lailafin Nahar Tithy**

**ID : 011201332**

**Submitted To**

Prof. Dr. Chowdhury Mofizur Rahman

**Submission Date: 03/05/2023**

Recurrent Neural Networks (RNNs) are a type of neural network that can process sequential data, such as time-series data or natural language. Training an RNN involves adjusting the weights of the network to minimize the error between the predicted outputs and the true outputs.

Let's take the example of training an **RNN for** **Sentiment Analysis on a dataset of Movie reviews.**

1. **Preprocessing the data:**

The first step is to preprocess the movie review data. This may involve tokenizing the reviews, converting the words to lowercase, removing stop words, and creating a vocabulary of all the unique words in the reviews.

1. **Defining the model architecture:**

For sentiment analysis, we can use an RNN with LSTM (Long Short-Term Memory) cells. The LSTM cells help the network to remember information from previous time steps, which is important for processing sequential data. We can define a model with one LSTM layer with 128 neurons, followed by a dense layer with a single neuron and a sigmoid activation function.

1. **Initializing the model:**

Once the architecture is defined, the next step is to initialize the model. This involves randomly assigning weights to the neurons in the network.

We can use the method of Glorot :

**initializer = tf.keras.initializers.GlorotUniform()**

1. **Forward propagation:**

During forward propagation, we feed the preprocessed movie review data into the network. The LSTM cells process the input sequence one word at a time, and the output at each time step is passed to the next time step. The final output of the network is the predicted sentiment score for the input review.

**h(t) = activation\_function(Wxh \* x(t) + Whh \* h(t-1) + bh)**

**y(t) = activation\_function(Why \* h(t) + by)**

**where ,**

**h(t) = Current hidden layer**

**y(t) = Output**

**activation function = Sigmoid Function**

**Wxh = weight of input layer**

**Whh = weight of hidden layer**

**Why = weight of output layer**

**by, bh = output bias , hidden bias**

1. **Calculating the loss:**

Once the predicted sentiment score is calculated, the next step is to compute the loss. We can use **Binary Cross-entropy loss**, which measures the difference between the predicted sentiment score and the true sentiment score for each movie review.

**L(y, y') = -[y \* log(y') + (1-y) \* log(1-y')]**

**where:**

**L: binary cross entropy loss**

**y: the true label (0 or 1)**

**y’: predicted output**

1. **Backpropagation:**

During backpropagation, the error is propagated backwards through the network, and the gradients of the weights are computed. These gradients are used to update the weights in the network.

1. **Updating the weights:**

The final step is to update the weights of the network using an optimization algorithm such as Adam. This involves adjusting the weights in the direction that minimizes the loss by using the following equation.

**W(new) = W(old) + learning\_rate \* gradient**

1. **Repeat steps 4-7:**

Steps 4-7 are repeated for a fixed number of epochs or until the loss converges to a minimum value. After training is complete, the model can be used to predict the sentiment of new movie reviews.

This is just a simplified example, and the actual implementation of an RNN for sentiment analysis would involve many additional steps, such as splitting the data into training and validation sets, using embeddings to represent the words, and tuning the hyperparameters of the model.