ML-LAB-13-Activities

1. Understand the **RNN architecture** and **LSTM model** (refer to material shared)
2. Demonstrate **text classification** (**binary**) for sentiment analysis on IMDB/similar dataset.
   i) Using SimpleRNN
   ii) Using LSTM
   iii) Using Bidirectional LSTM
3. Compare the accuracies of the above 3 methods for different parameters.

****

Prepare the Dataset:

Use a corpus with sentences and their corresponding POS tags.
Preprocess the dataset to tokenize sentences and encode POS tags.

Define the RNN Model Architecture:

Build an RNN model with LSTM or GRU layers.
The input will be sequences of word embeddings, and the output will be POS tag predictions.

Compile the Model:

Specify appropriate loss function and optimizer for multi-class classification.
Choose a suitable evaluation metric such as accuracy.

Train the Model:

Train the model on the training dataset.
Validate the model on a separate validation dataset.

Evaluate the Model:

Evaluate the trained model on a test dataset.
Measure accuracy or other relevant metrics to assess model performance.

****

Train a sentiment analysis model on IMDB dataset, use RNN layers with LSTM:

Load the IMDB dataset: The first step is to load the IMDB dataset, which contains movie reviews labeled as positive or negative.
Preprocess the data: Preprocess the text by removing stop words, special characters, and converting them into lowercase. Tokenize the text data into sequences of integers, and pad the sequences to a fixed length.
Split the data: Split the preprocessed data into training and testing sets.

Build the model: Build a sequential model in Keras, with an Embedding layer that will learn word embeddings from the data, followed by a RNN layer with LSTM nodes, and a dense layer with a sigmoid activation function that will output the sentiment probability.
Compile the model: Compile the model with binary cross-entropy as the loss function and Adam optimizer.
Train the model: Train the model on the training set, and validate it on the testing set. You can also use early stopping to prevent overfitting.
Evaluate the model: Evaluate the model on the testing set using metrics such as accuracy, precision, recall, and F1 score.
Make predictions: Use the trained model to make predictions on new text data.
Here is some sample code that can be used to train the sentiment analysis model on the IMDB dataset:
****

Implement Simple RNN for sentiment analysis on movie reviews.

```
# RNN sentiment analysis on movie reviews
import tensorflow as tf
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.datasets import imdb

# Configuration
max_features = 10000  # Vocabulary size
maxlen = 130          # Max length of a review
embedding_dim = 64    # Embedding dimensions

# 1. Load and Pad Data
(x_train, y_train), (x_test, y_test) =
imdb.load_data(num_words=max_features)
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)

# 2. Build Model
model = Sequential() #
model.add(Embedding(max_features, embedding_dim, input_length=maxlen))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))

# 3. Compile and Train
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=32, epochs=5,
validation_data=(x_test, y_test))

# 4. Evaluate
score, acc = model.evaluate(x_test, y_test, batch_size=32) #
print('Test accuracy:', acc)
```

****

Implement LSTM RNN for sentiment analysis on movie reviews

```
import numpy as np
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM
from keras_preprocessing import sequence
import matplotlib.pyplot as plt

# --- Configuration ---
max_features = 10000  # Vocabulary size (top most frequent words)
maxlen = 100          # Cut reviews after this many words (among top
max_features)
batch_size = 32
epochs = 5

# --- Data Preparation ---
print('Loading data...')
# The dataset is already preprocessed into sequences of integers (word
indices)
(x_train, y_train), (x_test, y_test) =
imdb.load_data(num_words=max_features)
print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')

print('Pad sequences (samples x time)...')
# Pad sequences to the same length
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)

# --- Model Building ---
print('Build model...')
model = Sequential()
# Embedding layer turns positive integers (word indices) into dense
vectors of fixed size
model.add(Embedding(max_features, 128, input_length=maxlen))
# LSTM layer with dropout to prevent overfitting
model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.2))
# Output layer with a single neuron and sigmoid activation for binary
classification (positive/negative)
model.add(Dense(1, activation='sigmoid'))

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

print(model.summary())

# --- Model Training ---
print('Train...')
history = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_data=(x_test, y_test),
          verbose=1)

# --- Model Evaluation ---
score, acc = model.evaluate(x_test, y_test,
                            batch_size=batch_size, verbose=0)
print('Test score:', score)
print('Test accuracy:', acc)

# --- Making Predictions ---
# Example prediction function
def predict_sentiment(text_review):
    # Tokenize and pad the input text to match the training data format
    # Note: A pre-fitted tokenizer would be needed here for custom input.
    # For this example, let's assume a function that handles this or use
existing test data.
    # The IMDb dataset is already numerical, so this part assumes a
process to convert raw text to padded sequence of indices.
        # Example with pre-processed test data
    sample_review = x_test[0:1] # Get the first test review
    prediction = (model.predict(sample_review) > 0.5).astype("int32") #
Use >0.5 for binary classification
    sentiment_label = "Positive" if prediction == 1 else "Negative"
    print(f"\nSample Review Sentiment: {sentiment_label}")

predict_sentiment(None) # Call the example prediction
```

****

Implement BiLSTM RNN for sentiment analysis on movie reviews

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

# Configuration and Data Preparation
max_features = 20000
maxlen = 100
(X_train, y_train), (X_test, y_test) =
imdb.load_data(num_words=max_features)
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)

# Model Building with Bidirectional LSTM
model = Sequential([
    Embedding(max_features, 128, input_length=maxlen),
    Bidirectional(LSTM(64)),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
```

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

# Training and Evaluation
model.fit(X_train, y_train, batch_size=32, epochs=3,
validation_data=(X_test, y_test))
score, acc = model.evaluate(X_test, y_test, batch_size=32)
print('Test accuracy:', acc)
**************************************************************************
```