Lab 16 : Design of LSTM and GRU RNN for classification od IMDB reviews:

Name : ASHIKA C

Roll No : 225229105

Step 1 : Imports : ¶

In [1]:
```
import numpy as np
import nltk
import re
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.datasets import imdb
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
```

In [2]:
```
max_words = 10000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_words)
```

Step 2 : Pre-processing the Text

In [3]:
```
stop_words = set(stopwords.words('english'))
```

In [4]:
```
def preprocess_text(text):
    # Convert to lowercase
    text = text.lower()

    # Remove special characters and numbers
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Tokenize and remove stopwords
    tokens = word_tokenize(text)
    tokens = [word for word in tokens if word not in stop_words]

    return ' '.join(tokens)
```

In [7]:
```
# Apply preprocessing to each sentence
x_train_processed = [preprocess_text(' '.join(map(str, review))) for review in :
x_test_processed = [preprocess_text(' '.join(map(str, review))) for review in x_

# Tokenize the text data
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(x_train_processed)
```

In [8]:
```
# Convert text data to sequences of integers
x_train_sequences = tokenizer.texts_to_sequences(x_train_processed)
x_test_sequences = tokenizer.texts_to_sequences(x_test_processed)

# Pad sequences to ensure they have the same length
max_sequence_length = 200  # Define the maximum sequence length
x_train_padded = pad_sequences(x_train_sequences, maxlen=max_sequence_length, pa
x_test_padded = pad_sequences(x_test_sequences, maxlen=max_sequence_length, pad
```

In [9]:
```
x_train_padded
```

Out[9]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

In [10]:
```
x_test_padded
```

Out[10]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

Step 3 : Dataset Preparation

In [11]:
```
# Split the data into training and testing sets (60% train, 40% test)
x_train, x_val, y_train, y_val = train_test_split(x_train_padded, y_train, test_

# Print the shapes of the sets to verify
print(f"x_train shape: {x_train.shape}")
print(f"x_val shape: {x_val.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_val shape: {y_val.shape}")
```

```
x_train shape: (15000, 200)
x_val shape: (10000, 200)
y_train shape: (15000,)
y_val shape: (10000,)
```

Step 4 : Model Creation

In [33]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, GlobalMaxPooling1D,
```

In [13]:
```
# Define the model
model = Sequential([
    # Embedding layer
    Embedding(input_dim=max_words, output_dim=128, input_length=max_sequence_le

    # LSTM layer
    LSTM(64),

    # Dense layer
    Dense(64, activation='relu'),

    # Output layer
    Dense(1, activation='sigmoid')
])
```

In [14]:
```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'

# Train the model for 10 epochs
model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=10, batch_si
```

```
Epoch 1/10
118/118 [==============================] - 23s 158ms/step - loss: 0.6934 - a
ccuracy: 0.5037 - val_loss: 0.6940 - val_accuracy: 0.5035
Epoch 2/10
118/118 [==============================] - 18s 153ms/step - loss: 0.6935 - a
ccuracy: 0.4993 - val_loss: 0.6935 - val_accuracy: 0.4965
Epoch 3/10
118/118 [==============================] - 18s 154ms/step - loss: 0.6934 - a
ccuracy: 0.4963 - val_loss: 0.6936 - val_accuracy: 0.4965
Epoch 4/10
118/118 [==============================] - 18s 151ms/step - loss: 0.6935 - a
ccuracy: 0.4934 - val_loss: 0.6937 - val_accuracy: 0.4965
Epoch 5/10
118/118 [==============================] - 19s 158ms/step - loss: 0.6933 - a
ccuracy: 0.4982 - val_loss: 0.6933 - val_accuracy: 0.4965
Epoch 6/10
118/118 [==============================] - 19s 158ms/step - loss: 0.6932 - a
ccuracy: 0.4947 - val_loss: 0.6931 - val_accuracy: 0.5035
Epoch 7/10
```

In [15]:
```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test_padded, y_test, verbose=0)
print(f'Test accuracy: {test_accuracy * 100:.2f}%')
```

```
Test accuracy: 50.00%
```

Step 5 : Run with Number of Layers

In [16]:
```
# Define the model
model1 = Sequential()
```

In [17]:
```
# Add an embedding layer
embedding_dim = 128
model1.add(Embedding(input_dim=max_words, output_dim=embedding_dim, input_lengt
```

In [18]:
```
# Add LSTM layers
num_units = [64, 128, 256]  # You can choose from 2, 3, or 4 layers

for units in num_units:
    model1.add(LSTM(units, return_sequences=True, dropout=0.2))
```

In [26]:
```
# Add Global Max Pooling layer
model1.add(GlobalMaxPooling1D())
```

In [27]:
```
# Add Dense layers
model1.add(Dense(64, activation='relu'))
model1.add(Dense(1, activation='sigmoid'))
```

In [28]:
```
# Compile the model
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy
```

In [30]:
```
# Train the model
epochs = 5
model1.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=epochs, bat
```

```
Epoch 1/5
118/118 [==============================] - 125s 1s/step - loss: 0.6932 - accu
racy: 0.4967 - val_loss: 0.6932 - val_accuracy: 0.4965
Epoch 2/5
118/118 [==============================] - 123s 1s/step - loss: 0.6932 - accu
racy: 0.4999 - val_loss: 0.6933 - val_accuracy: 0.4965
Epoch 3/5
118/118 [==============================] - 126s 1s/step - loss: 0.6932 - accu
racy: 0.4974 - val_loss: 0.6931 - val_accuracy: 0.5035
Epoch 4/5
118/118 [==============================] - 130s 1s/step - loss: 0.6932 - accu
racy: 0.5029 - val_loss: 0.6932 - val_accuracy: 0.4965
Epoch 5/5
118/118 [==============================] - 126s 1s/step - loss: 0.6932 - accu
racy: 0.4994 - val_loss: 0.6931 - val_accuracy: 0.5035
```

Out[30]:
```
<keras.callbacks.History at 0x1981f79e170>
```

In [31]:
```
# Evaluate the model on the test set
test_loss, test_accuracy = model1.evaluate(x_test_padded, y_test, verbose=0)
print(f'Test accuracy: {test_accuracy * 100:.2f}%')
```

```
Test accuracy: 50.00%
```

Step 6 : Variations:

Bidirectional LSTM Model :

In [34]:
```
# Define the model
model2 = Sequential()

# Add an embedding layer
embedding_dim = 128
model2.add(Embedding(input_dim=max_words, output_dim=embedding_dim, input_lengt

# Add Bidirectional LSTM layer
model2.add(Bidirectional(LSTM(64, return_sequences=True, dropout=0.2)))

# Add a Dense layer
model2.add(Dense(64, activation='relu'))

# Add the output Dense layer
model2.add(Dense(1, activation='sigmoid'))
```

In [36]:
```
# Add Global Max Pooling layer
model2.add(GlobalMaxPooling1D())
```

In [37]:
```
# Compile the model
model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy

# Train the model
epochs = 5
model2.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=epochs, bat
```

```
Epoch 1/5
118/118 [==============================] - 49s 368ms/step - loss: 0.6934 - ac
curacy: 0.4967 - val_loss: 0.6933 - val_accuracy: 0.4965
Epoch 2/5
118/118 [==============================] - 42s 357ms/step - loss: 0.6934 - ac
curacy: 0.5008 - val_loss: 0.6931 - val_accuracy: 0.5035
Epoch 3/5
118/118 [==============================] - 42s 356ms/step - loss: 0.6934 - ac
curacy: 0.4957 - val_loss: 0.6931 - val_accuracy: 0.5035
Epoch 4/5
118/118 [==============================] - 43s 361ms/step - loss: 0.6932 - ac
curacy: 0.5021 - val_loss: 0.6931 - val_accuracy: 0.5035
Epoch 5/5
118/118 [==============================] - 42s 360ms/step - loss: 0.6932 - ac
curacy: 0.4979 - val_loss: 0.6933 - val_accuracy: 0.4965
```

Out[37]:
```
<keras.callbacks.History at 0x1981f759ea0>
```

In [ ]:
```
# Evaluate the model on the test set
test_loss, test_acc = model2.evaluate(x_test_padded, y_test)
print(f'Test accuracy: {test_acc * 100:.2f}%')
```

Compare the Performance of LSTM and Bidirectional LSTM Models :

LSTM Accuracy Value : 50.00%

Bidirectional Accuracy Value :

Different Sequence Lengths :

In [ ]:
```
# Pad sequences to ensure they have the same length
max_sequence_length = 300  # Define the maximum sequence length
x_train_padded = pad_sequences(x_train_sequences, maxlen=max_sequence_length, pa
x_test_padded = pad_sequences(x_test_sequences, maxlen=max_sequence_length, padd
```

In [ ]:
```
# Split the data into training and testing sets (60% train, 40% test)
x_train, x_val, y_train, y_val = train_test_split(x_train_padded, y_train, test_

# Print the shapes of the sets to verify
print(f"x_train shape: {x_train.shape}")
print(f"x_val shape: {x_val.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_val shape: {y_val.shape}")
```

In [ ]:
```
# Define the model
model3 = Sequential()

# Add an embedding layer
embedding_dim = 128
model3.add(Embedding(input_dim=max_words, output_dim=embedding_dim, input_lengt

# Add Bidirectional LSTM layer
model3.add(Bidirectional(LSTM(64, return_sequences=True, dropout=0.2)))

# Add a Dense layer
model3.add(Dense(64, activation='relu'))

# Add the output Dense layer
model3.add(Dense(1, activation='sigmoid'))
```

In [ ]:
```
# Compile the model
model3.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy

# Train the model
epochs = 5
model3.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=epochs, bat
```

In [ ]:
```
# Evaluate the model on the test set
test_loss, test_acc = model3.evaluate(x_test_padded, y_test)
print(f'Test accuracy: {test_acc * 100:.2f}%')
```

User dropouts and observe the performance :

In [ ]:
```
# Define the model
model4 = Sequential()

# Add an embedding layer
embedding_dim = 128
model4.add(Embedding(input_dim=max_words, output_dim=embedding_dim, input_lengt

# Add Bidirectional LSTM layer
model4.add(Bidirectional(LSTM(64, return_sequences=True, dropout=0.2)))

# Add a Dense layer with dropout
model4.add(Dense(64, activation='relu'))
model4.add(Dropout(0.5))

# Add the output Dense layer
model4.add(Dense(1, activation='sigmoid'))
```

In [ ]:
```
# Compile the model
model4.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy

# Train the model
epochs = 5
model4.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=epochs, bat
```

In [ ]: