Annapoornima S

225229101

PDL Lab17. Text Classification using CNN-LSTM and Pre-trained Glove Models

In [1]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, LS
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.datasets import fetch_20newsgroups
import tensorflow_datasets as tfds
```

In [2]:
```
dataset, info = tfds.load('ag_news_subset', split='train', with_info=Tr
```

```
Downloading and preparing dataset 11.24 MiB (download: 11.24 MiB, gen
erated: 35.79 MiB, total: 47.03 MiB) to /root/tensorflow_datasets/ag_
news_subset/1.0.0...

Dl Completed...: 0 url [00:00, ? url/s]

Dl Size...: 0 MiB [00:00, ? MiB/s]

Extraction completed...: 0 file [00:00, ? file/s]

Generating splits...:   0%|          | 0/2 [00:00<?, ? splits/s]

Generating train examples...:   0%|          | 0/120000 [00:00<?, ? e
xamples/s]

Shuffling /root/tensorflow_datasets/ag_news_subset/1.0.0.incomplete8S
O6X2/ag_news_subset-train.tfrecord*...:  …

Generating test examples...:   0%|          | 0/7600 [00:00<?, ? exam
ples/s]

Shuffling /root/tensorflow_datasets/ag_news_subset/1.0.0.incomplete8S
O6X2/ag_news_subset-test.tfrecord*...:   …

Dataset ag_news_subset downloaded and prepared to /root/tensorflow_da
tasets/ag_news_subset/1.0.0. Subsequent calls will reuse this data.
```

In [ ]:
```
# Extract text and labels from the dataset
X = [example['description'].numpy().decode('utf-8') for example in data
y = [example['label'].numpy() for example in dataset]
```

In [5]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
```

In [6]:
```
# Tokenize the text data
max_words = 10000
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X_train)
```

In [7]:
```
# Convert text data to sequences and pad them to a fixed length
max_sequence_length = 200
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)
X_train_padded = pad_sequences(X_train_seq, maxlen=max_sequence_length)
X_test_padded = pad_sequences(X_test_seq, maxlen=max_sequence_length)
```

In [8]:
```
# Load pre-trained GloVe embeddings
embedding_dim = 100
embedding_index = {}
glove_file = 'glove.6B.100d.txt'
```

In [9]:
```
with open(glove_file, encoding="utf8") as f:
    for line in f:
        word, vec = line.split(maxsplit=1)
        embedding_index[word] = np.array([float(val) for val in vec.spl
```

In [10]:
```
# Create the embedding matrix
word_index = tokenizer.word_index
embedding_matrix = np.zeros((max_words, embedding_dim))
```

In [11]:
```
for word, i in word_index.items():
    if i < max_words:
        embedding_vector = embedding_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

In [13]:
```
# Calculate the number of classes
num_classes = len(np.unique(y))

# Create the CNN-LSTM model with GloVe embeddings
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=max_sequence
model.add(Conv1D(128, 5, activation='relu'))
model.add(MaxPooling1D(5))
model.add(LSTM(128))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

In [14]:
```
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001)

# One-hot encode the labels
y_train_onehot = to_categorical(y_train, num_classes)
y_test_onehot = to_categorical(y_test, num_classes)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learn
ing_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legac
y.Adam.
```

In [15]:
```
# Define early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_
```

In [16]:
```
# Train the model
epochs = 10
batch_size = 64
history = model.fit(X_train_padded, y_train_onehot, epochs=epochs, batc
```

```
Epoch 1/10
1200/1200 [==============================] - 23s 8ms/step - loss: 0.3
968 - accuracy: 0.8636 - val_loss: 0.3360 - val_accuracy: 0.8837
Epoch 2/10
1200/1200 [==============================] - 9s 8ms/step - loss: 0.30
74 - accuracy: 0.8939 - val_loss: 0.3076 - val_accuracy: 0.8886
Epoch 3/10
1200/1200 [==============================] - 9s 7ms/step - loss: 0.26
55 - accuracy: 0.9075 - val_loss: 0.3075 - val_accuracy: 0.8935
Epoch 4/10
1200/1200 [==============================] - 8s 7ms/step - loss: 0.22
90 - accuracy: 0.9198 - val_loss: 0.3021 - val_accuracy: 0.8977
Epoch 5/10
1200/1200 [==============================] - 9s 8ms/step - loss: 0.19
72 - accuracy: 0.9308 - val_loss: 0.3341 - val_accuracy: 0.8896
Epoch 6/10
1200/1200 [==============================] - 9s 8ms/step - loss: 0.16
79 - accuracy: 0.9407 - val_loss: 0.3486 - val_accuracy: 0.8898
Epoch 7/10
1200/1200 [==============================] - 8s 7ms/step - loss: 0.14
28 - accuracy: 0.9492 - val_loss: 0.3448 - val_accuracy: 0.8961
Epoch 8/10
1200/1200 [==============================] - 9s 7ms/step - loss: 0.11
78 - accuracy: 0.9581 - val_loss: 0.4140 - val_accuracy: 0.8936
Epoch 9/10
1200/1200 [==============================] - 9s 7ms/step - loss: 0.09
91 - accuracy: 0.9634 - val_loss: 0.4659 - val_accuracy: 0.8922
```

In [17]:
```
# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test_padded, y_test_onehot)
print(f'Test Loss: {loss:.4f}')
print(f'Test Accuracy: {accuracy:.4f}')
```

```
750/750 [==============================] - 4s 5ms/step - loss: 0.2906
- accuracy: 0.8993
Test Loss: 0.2906
Test Accuracy: 0.8993
```