LSTM-Glove model

1 - Import Data

In [ ]:
```
import json
import numpy as np
import pandas as pd

emotion = pd.read_csv('/kaggle/input/dm-2024-isa-5810-lab-2-homework/emotior
data_identification = pd.read_csv('/kaggle/input/dm-2024-isa-5810-lab-2-home

# import and convert tweet (a bit slow, could be optimized)
with open('/kaggle/input/dm-2024-isa-5810-lab-2-homework/tweets_DM.json', 'r
        data = [json.loads(line) for line in f]

df = pd.DataFrame(data)
_source = df['_source'].apply(lambda x: x['tweet'])
df = pd.DataFrame({
    'tweet_id': _source.apply(lambda x: x['tweet_id']),
    'text': _source.apply(lambda x: x['text']),
})
df = df.merge(data_identification, on='tweet_id', how='left')
```

In [ ]:
```
# train test split
test_data = df[df['identification'] == 'test']

train_data = df[df['identification'] == 'train']
train_data = train_data.merge(emotion, on='tweet_id', how='left')
```

2 - Setup TPU

In [ ]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

try:
    # Detect TPU
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    # Connect to TPU cluster
    tf.config.experimental_connect_to_cluster(tpu)
    # Initialize TPU system
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
except ValueError:
    tpu = None
    strategy = tf.distribute.get_strategy()
```

3 - Preprocessing

In [ ]:
```
import nltk
import re
from nltk.corpus import stopwords

nltk.download('stopwords')
nltk.download('punkt')
nltk.download('punkt_tab')
```

3.1 - Oversample emotions to balance classes

In [ ]:
```
from sklearn.utils import resample
import pandas as pd

# Hacky way to oversample
def oversample_emotions(train_data):
    emotion_counts = train_data['emotion'].value_counts()

    max_count = emotion_counts.max()

    oversampled_dfs = []


    for emotion in emotion_counts.index:
        emotion_df = train_data[train_data['emotion'] == emotion]

        oversampled = resample(emotion_df,
                               replace=True,
                               n_samples=max_count,
                               random_state=0) # NON DETERMINISTIC

        oversampled_dfs.append(oversampled)

    balanced_data = pd.concat(oversampled_dfs)

    return balanced_data


balanced_train_data = oversample_emotions(train_data)
train_data = balanced_train_data
```

3.2 - Text preprocessing

Clean and tokenize text by removing URLs, mentions, hashtags, special characters, numbers, and stop words, while converting the text to lowercase.

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

def clean_text(text):
    text = re.sub(r'http\S+', '', text) # URLs
    text = re.sub(r'@\w+|#\w+', '', text) # '@'s and '#'s
    text = re.sub(r'[^A-Za-z\s]', '', text)

    text = text.lower()
```

```
    tokens = nltk.word_tokenize(text)

    tokens = [word for word in tokens if word not in stop_words] # Stop word
    return tokens

train_data['tokens'] = train_data['text'].apply(clean_text)
test_data['tokens'] = test_data['text'].apply(clean_text)

train_data.head()
```

In [ ]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer()
tokenizer.fit_on_texts(train_data['tokens'])
sequences = tokenizer.texts_to_sequences(train_data['tokens'])
word_index = tokenizer.word_index

max_length = max(len(seq) for seq in sequences)
padded_sequences = pad_sequences(sequences, maxlen=max_length, padding='post
```

3.3 - Class Encoding

In [ ]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
train_data["emotion_enc"] = le.fit_transform(train_data["emotion"])
```

3.4 Embeddings

In [ ]:
```
embeddings_index = {}
with open('/kaggle/input/glove-6b-100d-txt/glove.6B.100d.txt', encoding='utf
    for line in f:
        values = line.split()
        word = values[0]
        coeffs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coeffs
```

In [ ]:
```
embedding_dim = 100
vocab_size = len(tokenizer.word_index) + 1

embedding_matrix = np.zeros((vocab_size, embedding_dim))

for word, i in tokenizer.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

4 - Model

Using a RNN (Long Short Term Memory), in order to extract more context

In [ ]:
```
with strategy.scope():
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

    model = Sequential()
    model.add(Embedding(input_dim=vocab_size,
                        output_dim=embedding_dim,
                        weights=[embedding_matrix],
                        trainable=True))
    model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))

    num_classes = len(le.classes_)
    if num_classes > 2:
        model.add(Dense(num_classes, activation='softmax'))
    else:
        model.add(Dense(1, activation='sigmoid'))
```

In [ ]:
```
with strategy.scope():
    if num_classes > 2:
        model.compile(loss='sparse_categorical_crossentropy',
                      optimizer='adam',
                      metrics=['accuracy'])
    else:
        model.compile(loss='binary_crossentropy',
                      optimizer='adam',
                      metrics=['accuracy'])
```

In [ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(padded_sequences, train_da
```

In [ ]:
```
batch_size = 16 * strategy.num_replicas_in_sync
```

5 - Prediction

In [ ]:
```
import numpy as np

new_sequences = tokenizer.texts_to_sequences(test_data['tokens'])
new_X = pad_sequences(new_sequences, maxlen=128)
predictions = model.predict(new_X)

predicted_classes = np.argmax(predictions, axis=1)
predicted_emotions = le.inverse_transform(predicted_classes)
```

In [ ]:
```
submission = pd.DataFrame({
    'id': test_data['tweet_id'],
    'emotion': predicted_emotions
})

submission = submission[['id', 'emotion']]
submission.to_csv('submission.csv', index=False)
```

```
print("HOoray")
```

6 - Observations

This method performs much poorly than Bert, which is to be expected but the upside is that it is able to train on TPUs and is much faster.

In [ ]: