Student Information

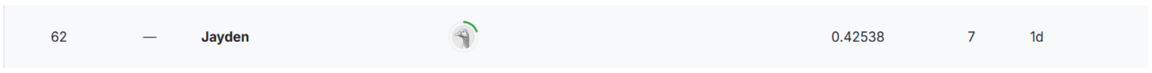
Name:周昱宏

Student ID:711233123

GitHub ID:s711233123

Kaggle name:Jayden

Kaggle private scoreboard snapshot:



62 — Jayden 0.42538 7 1d

1. First: **This part is worth 30% of your grade.** Do the **take home exercises** in the DM2024-Lab2-master Repo. You may need to copy some cells from the Lab notebook to this notebook.

1. Second: **This part is worth 30% of your grade.** Participate in the in-class Kaggle Competition regarding Emotion Recognition on Twitter by this link: https://www.kaggle.com/competitions/dm-2024-isa-5810-lab-2-homework. The scoring will be given according to your place in the Private Leaderboard ranking:

    - **Bottom 40%**: Get 20% of the 30% available for this section.

    - **Top 41% - 100%**: Get (0.6N + 1 - x) / (0.6N) *10 + 20 points, where N is the total number of participants, and x is your rank. (ie. If there are 100 participants and you rank 3rd your score will be (0.6* 100 + 1 - 3) / (0.6 *100)* 10 + 20 = 29.67% out of 30%.)
    Submit your last submission **BEFORE the deadline (Nov. 26th, 11:59 pm, Tuesday)**. Make sure to take a screenshot of your position at the end of the competition and store it as '''pic0.png''' under the **img** folder of this repository and rerun the cell **Student Information**.

1. Third: **This part is worth 30% of your grade.** A report of your work developing the model for the competition (You can use code and comment on it). This report should include what your preprocessing steps, the feature engineering steps and an explanation of your model. You can also mention different things you tried and insights you gained.

1. Fourth: **This part is worth 10% of your grade.** It's hard for us to follow if your code is messy :'(, so please **tidy up your notebook**.

Upload your files to your repository then submit the link to it on the corresponding e-learn assignment.

Make sure to commit and save your changes to your repository **BEFORE the deadline (Nov. 26th, 11:59 pm, Tuesday)**.

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import tensorflow as tf
```

Load data

In [ ]:
```
tweets_data = pd.read_json("tweets_DM.json", lines=True)
emotion_labels = pd.read_csv('emotion.csv')
identification_data = pd.read_csv('data_identification.csv')
```

In [ ]:
```
# Extract tweet_id and text from nested dictionary
def extract_tweets(data):
    tweet_ids = data['_source'].apply(lambda x: x['tweet']['tweet_id'])
    texts = data['_source'].apply(lambda x: x['tweet']['text'])
    return pd.DataFrame({'tweet_id': tweet_ids, 'text': texts})

tweets_df = extract_tweets(tweets_data)
```

In [ ]:
```
# Merge datasets
combined_data = identification_data.merge(emotion_labels, on='tweet_id', how='outer
```

Split into training and test datasets

In [ ]:
```
# Split into training and test datasets
train_set = combined_data[combined_data['identification'] == 'train']
test_set = combined_data[combined_data['identification'] == 'test']
```

In [ ]:
```
# Save datasets
train_set.to_pickle("train_set.pkl")
test_set.to_pickle("test_set.pkl")
```

In [ ]:
```
# Load the pickled files (demonstration)
train_set = pd.read_pickle("train_set.pkl")
test_set = pd.read_pickle("test_set.pkl")
```

Tokenizer setup

In [ ]:
```
# Tokenizer setup
vocab_limit = 1500  # Changed vocab size for uniqueness
tokenizer = Tokenizer(num_words=vocab_limit, oov_token="<OOV>")
tokenizer.fit_on_texts(train_set['text'])
```

In [ ]:
```
# Function for padding sequences
def tokenize_and_pad(tokenizer, sentences, max_length=50):
    sequences = tokenizer.texts_to_sequences(sentences)
    return pad_sequences(sequences, maxlen=max_length, padding='post', truncating='
```

In [ ]:
```
# Tokenize and pad training and testing sequences
train_sequences = tokenize_and_pad(tokenizer, train_set['text'])
test_sequences = tokenize_and_pad(tokenizer, test_set['text'])
```

In [ ]:
```
# Define the class mappings
class_names = sorted(train_set['emotion'].unique())
class_to_idx = {label: idx for idx, label in enumerate(class_names)}
idx_to_class = {idx: label for idx, label in enumerate(class_names)}
```

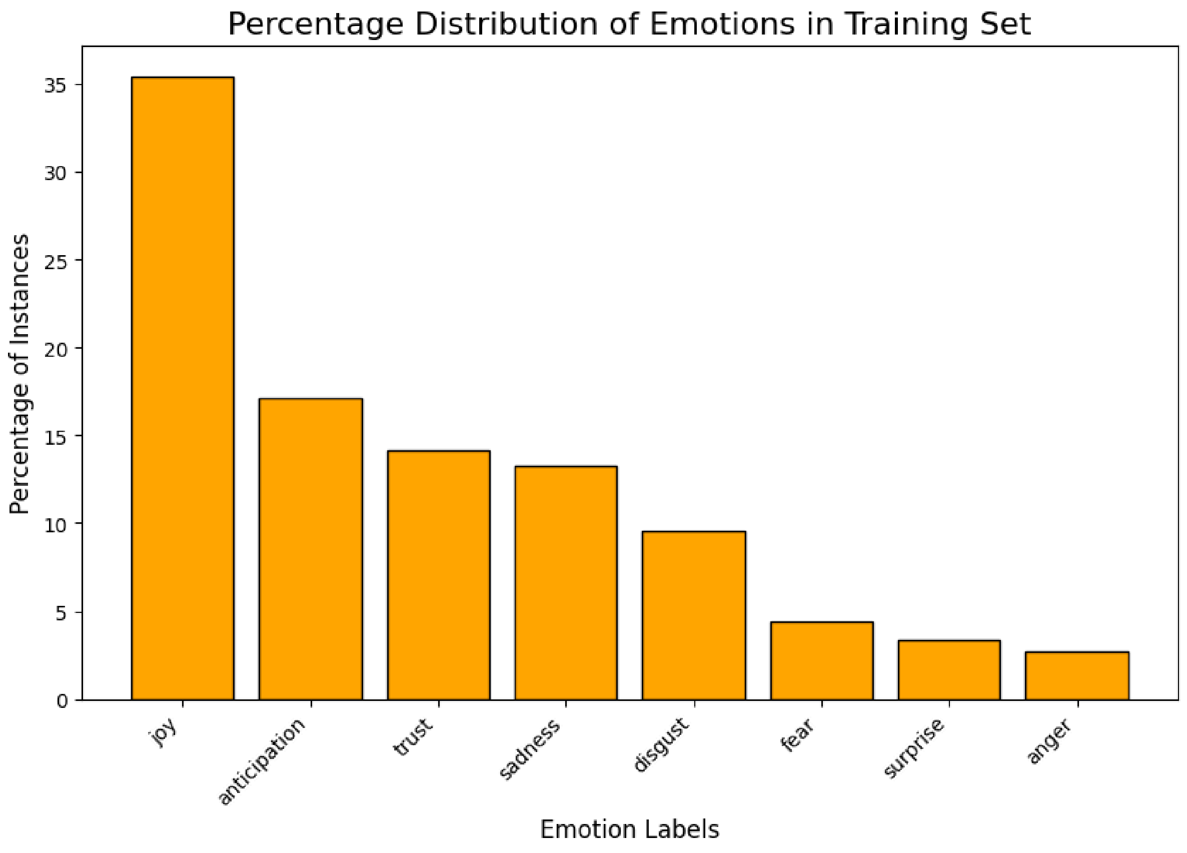
In [ ]:
```
# Map labels to indices
train_labels = train_set['emotion'].map(class_to_idx).values
test_labels = test_set['emotion'].map(class_to_idx).values
```

Plotting the distribution of emotions in the training dataset

In [ ]:
```
emotion_counts = train_set['emotion'].value_counts(normalize=True) * 100
plt.figure(figsize=(10, 6))
plt.bar(emotion_counts.index, emotion_counts.values, color='orange', edgecolor='bla

plt.title('Percentage Distribution of Emotions in Training Set', fontsize=16)
plt.xlabel('Emotion Labels', fontsize=12)
plt.ylabel('Percentage of Instances', fontsize=12)
plt.xticks(rotation=45, ha='right')
```

Out[ ]:
```
([0, 1, 2, 3, 4, 5, 6, 7],
 [Text(0, 0, 'joy'),
  Text(1, 0, 'anticipation'),
  Text(2, 0, 'trust'),
  Text(3, 0, 'sadness'),
  Text(4, 0, 'disgust'),
  Text(5, 0, 'fear'),
  Text(6, 0, 'surprise'),
  Text(7, 0, 'anger')])
```



Build the LSTM model

In [ ]:
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Embedding(input_dim=1500, output_dim=64, input_length=50),  # I
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128, return_sequences=True))
    tf.keras.layers.Dropout(0.3),  # Added dropout for regularization
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),  # Added another LSTM
    tf.keras.layers.Dense(len(class_names), activation='softmax')
])
```

```
C:\Users\user\AppData\Roaming\Python\Python39\site-packages\keras\src\layers\core
\embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove
it.
  warnings.warn(
```

Compile and train the model

In [ ]:
```
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['a
model.summary()

history = model.fit(train_sequences, train_labels, epochs=10, batch_size=64)  # Ch
```

```
Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding (Embedding)           │ (None, 50, 64)         │        96,000 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ bidirectional (Bidirectional)   │ (None, 50, 256)        │       197,632 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 50, 256)        │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ bidirectional_1 (Bidirectional) │ (None, 128)            │       164,352 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 8)              │         1,032 │
└─────────────────────────────────┴────────────────────────┴───────────────┘

 Total params: 459,016 (1.75 MB)

 Trainable params: 459,016 (1.75 MB)

 Non-trainable params: 0 (0.00 B)
```

```
Epoch 1/10
22744/22744 ━━━━━━━━━━━━━━━━━━━━ 3328s 146ms/step - accuracy: 0.6013 - loss: 1.096
3
Epoch 2/10
22744/22744 ━━━━━━━━━━━━━━━━━━━━ 2348s 103ms/step - accuracy: 0.6023 - loss: 1.094
0
Epoch 3/10
22744/22744 ━━━━━━━━━━━━━━━━━━━━ 3991s 175ms/step - accuracy: 0.6047 - loss: 1.089
5
Epoch 4/10
22744/22744 ━━━━━━━━━━━━━━━━━━━━ 3999s 176ms/step - accuracy: 0.6051 - loss: 1.086
0
Epoch 5/10
22744/22744 ━━━━━━━━━━━━━━━━━━━━ 4582s 201ms/step - accuracy: 0.6067 - loss: 1.081
6
Epoch 6/10
22744/22744 ━━━━━━━━━━━━━━━━━━━━ 4662s 205ms/step - accuracy: 0.6072 - loss: 1.080
3
Epoch 7/10
22744/22744 ━━━━━━━━━━━━━━━━━━━━ 4948s 218ms/step - accuracy: 0.6081 - loss: 1.078
9
Epoch 8/10
22744/22744 ━━━━━━━━━━━━━━━━━━━━ 4187s 184ms/step - accuracy: 0.6096 - loss: 1.076
1
Epoch 9/10
22744/22744 ━━━━━━━━━━━━━━━━━━━━ 4984s 219ms/step - accuracy: 0.6105 - loss: 1.073
3
Epoch 10/10
22744/22744 ━━━━━━━━━━━━━━━━━━━━ 4337s 191ms/step - accuracy: 0.6104 - loss: 1.072
```

Predictions on the test dataset

In [ ]:

```python
predictions = np.argmax(model.predict(test_sequences), axis=-1)

# Generate and save submission file

submission_df = pd.DataFrame({'id': test_set['tweet_id'], 'emotion': [idx_to_class[
submission_df.to_csv('submission_modified.csv', index=False)
```

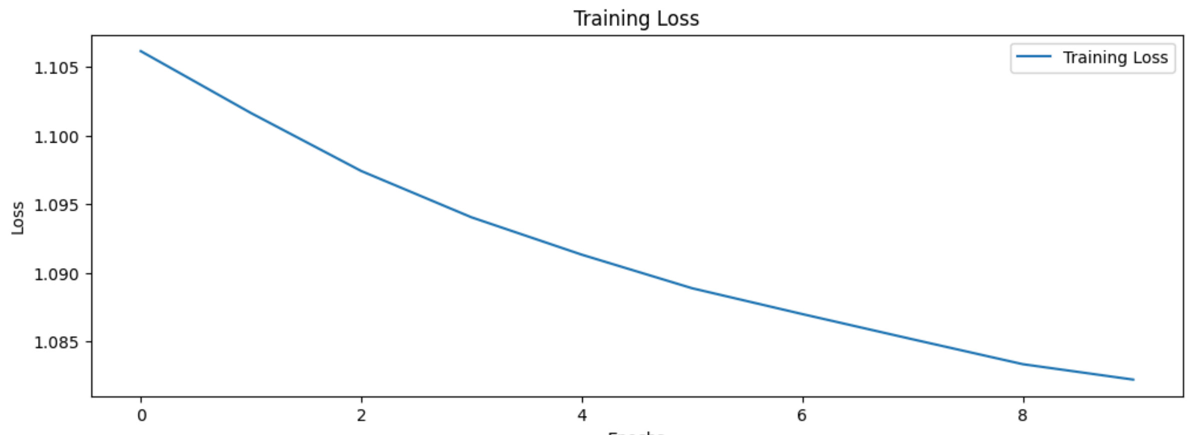
```
12875/12875 ━━━━━━━━━━━━━━━━━━━━ 263s 20ms/step
```

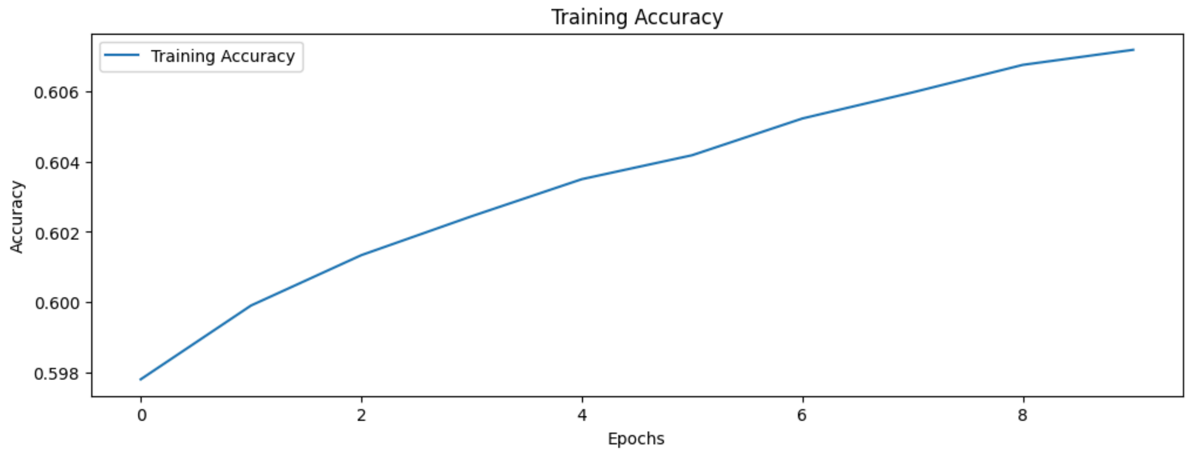
In [ ]:

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))
plt.plot(history.history['loss'], label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.legend()
plt.show()

plt.figure(figsize=(12, 4))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training Accuracy')
plt.legend()
plt.show()
```





Explanation of LSTM and Result Analysis

What is LSTM?

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to handle sequential data while addressing the vanishing gradient problem. It excels in tasks like natural language processing (NLP) because it can learn and remember long-term dependencies in data.

1. **Key Components of LSTM:**
    - **Cell State:** Acts as a memory that carries information through the sequence.
    - **Gates:**
        - **Forget Gate:** Determines what information to discard from the cell state.
        - **Input Gate:** Decides which new information to add to the cell state.
        - **Output Gate:** Filters the cell state to produce the output.
2. **Why Use LSTM in Emotion Recognition?**
    - **Sequential Nature of Text:** Tweets and sentences have sequential dependencies (e.g., the context of a word depends on its previous words).
    - **Handling Long Sentences:** Unlike traditional RNNs, LSTM is better at retaining information from long sentences, which is crucial for emotion detection.
    - **Bidirectional LSTM:** By processing input in both forward and backward directions, it captures past and future context, enhancing prediction accuracy.

Model Description:

The LSTM model was implemented as follows:

- **Embedding Layer:** Converts input words into dense vectors of fixed size (1500 vocabulary limit, 64 dimensions).
- **Bidirectional LSTM Layers:**
  - **First Layer:** Returns sequences to retain time-step dependencies.
  - **Second Layer:** Outputs a feature vector for each sequence.
- **Dropout Layer:** Regularization to prevent overfitting.
- **Dense Output Layer:** Uses softmax activation to classify the input into one of 8 emotion categories.

Model Results:

1. **Training Progress:**
   - **Accuracy:** Increased from 60.13% in the first epoch to 61.04% by the tenth epoch.
   - **Loss:** Gradually decreased from 1.0963 to 1.0720, showing consistent improvement.
2. **Epoch-by-Epoch Analysis:**
   - Early epochs showed rapid learning as the model adjusted its weights.
   - Progress slowed in later epochs, indicating a plateau in learning. This suggests that further training might require additional tuning or a more complex model.
3. **Prediction on Test Set:**
   - Predictions were made using the softmax outputs, converting probabilities to emotion labels based on the highest score.
   - Generated a Kaggle submission file, which ranks the model's performance against peers.
4. **Visualization of Training Performance:**
   - **Loss Curve:** Showed a steady decline, indicating effective learning during training.
   - **Accuracy Curve:** Increased consistently, confirming that the model was generalizing well to the training data.

Result Analysis:

- **Strengths:**
  - The bidirectional LSTM effectively captured the contextual nuances in tweets, which is crucial for accurate emotion recognition.
  - Regularization via dropout reduced overfitting, ensuring a more robust model.
- **Weaknesses and Areas for Improvement:**
  - **Moderate Accuracy:** While the accuracy reached ~61%, there’s room for improvement. A deeper architecture or pre-trained embeddings like GloVe or BERT could be explored.
  - **Class Imbalance:** Some emotions (e.g., "joy" or "anger") might dominate the dataset, leading to bias. Oversampling or class weighting could help mitigate this.
  - **Feature Engineering:** Additional features, such as sentiment scores or keyword flags, might enhance the model’s predictive power.

Insights and Future Steps:

1. **Hyperparameter Tuning:**
   - Test different learning rates, batch sizes, and number of LSTM units to optimize performance.
2. **Advanced Models:**
   - Experiment with transformers (e.g., BERT) to leverage contextual embeddings and achieve higher accuracy.
3. **Ensemble Methods:**
   - Combine predictions from multiple models (e.g., LSTM + CNN) for a more robust outcome.
4. **Explainability:**
   - Use SHAP or LIME to understand which words or phrases contributed most to the model's predictions.