Sequential Data

Presented to: Mr. Dani AZZAM – inmind .academy
Presented by: Hassan KHADRA – UA
Elias–Charbel SALAMEH – USEK

Agenda

01 – The Need for Sequential Models

02 – Applications

03 – What is RNN?

04 – Architecture

05 – Learning Process

06 – RNN Challenges

07 – The need to improve

08 – Long–Short Term Memory Neural Network

09 – Gated Recurrent Unit

10 – RNN x LSTM x GRU

11 – Dataset Chosen

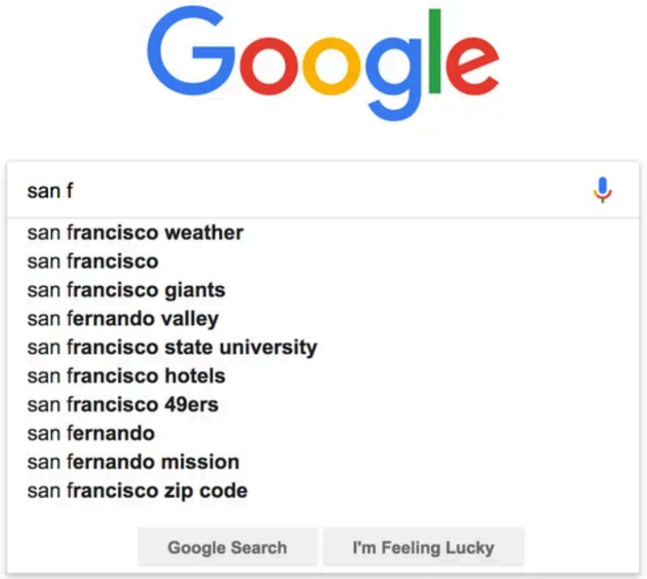
12– Code

The Need for Sequential Models

- Traditional models treat each data point as independent.
- The market lacks models that are capable of capturing depencies in sequential data.

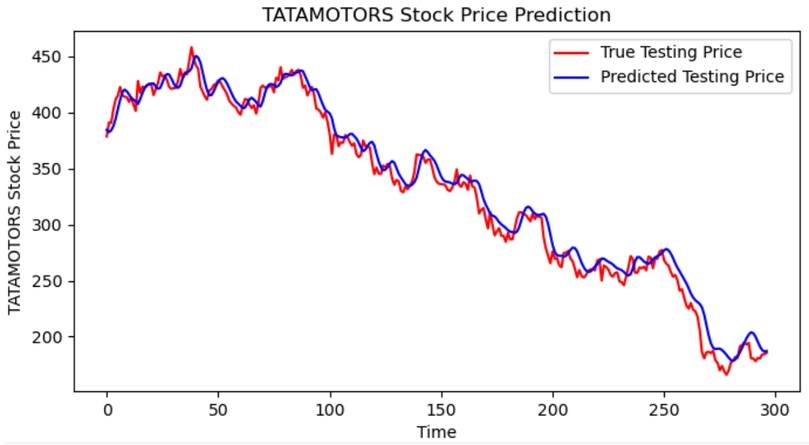
Applications

Text Generation



Speech Recognition

Time-Series Analysis



What is an RNN?

1. It has neurons that process data at each time step
2. The new output is affected by its current input as well as its previous output.
3. The key idea is that an RNN has memory.

Architecture

>>> <u>Basic RNN Structure:</u>

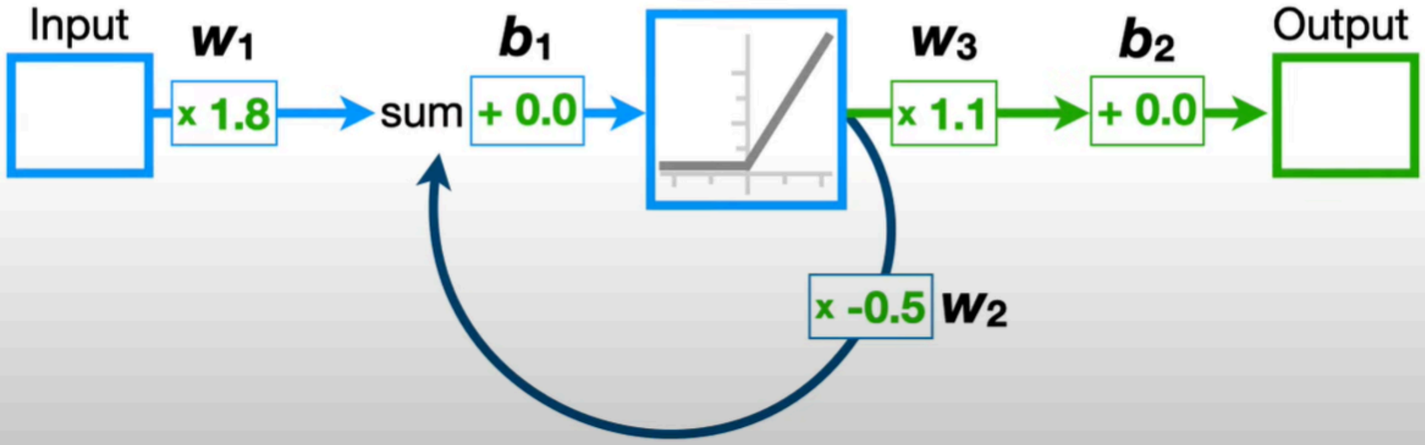
input layer - hidden layer - output layer

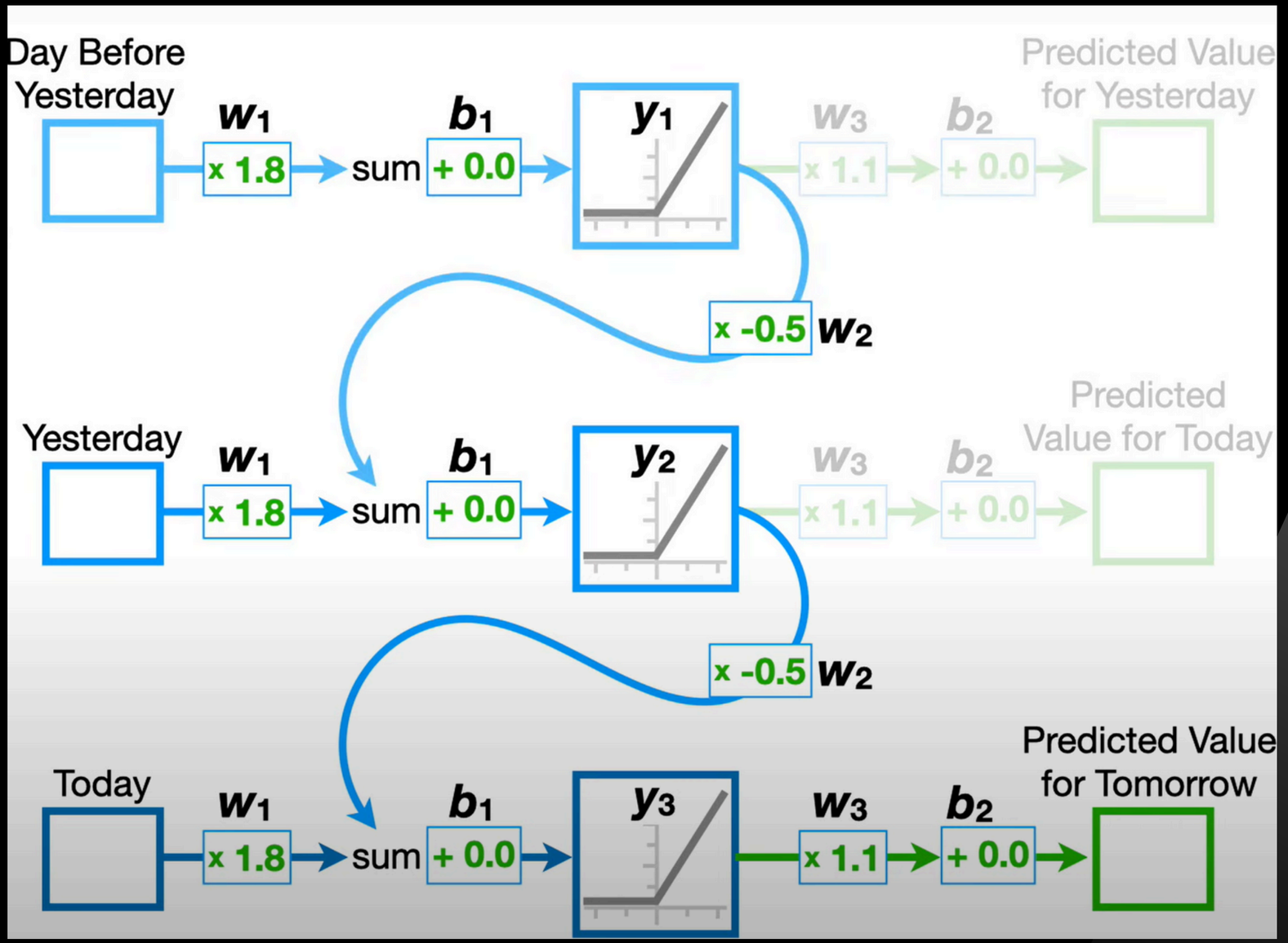
At each time step, the RNN processes the input and updates its hidden state.

>>> <u>Recurrent Connection:</u>

The model "remembers" past information because it’s affected by its previous output.

Just like the other neural networks that we've seen before, **Recurrent Neural Networks** have **weights, biases,**




Day Before Yesterday
w1
x 1.8
sum
b1
+ 0.0
y1
w3
x 1.1
b2
+ 0.0
Predicted Value for Yesterday
x -0.5
w2
Yesterday
w1
x 1.8
sum
b1
+ 0.0
y2
w3
x 1.1
b2
+ 0.0
Predicted Value for Today
x -0.5
w2
Today
w1
x 1.8
sum
b1
+ 0.0
y3
w3
x 1.1
b2
+ 0.0
Predicted Value for Tomorrow

The Learning Process

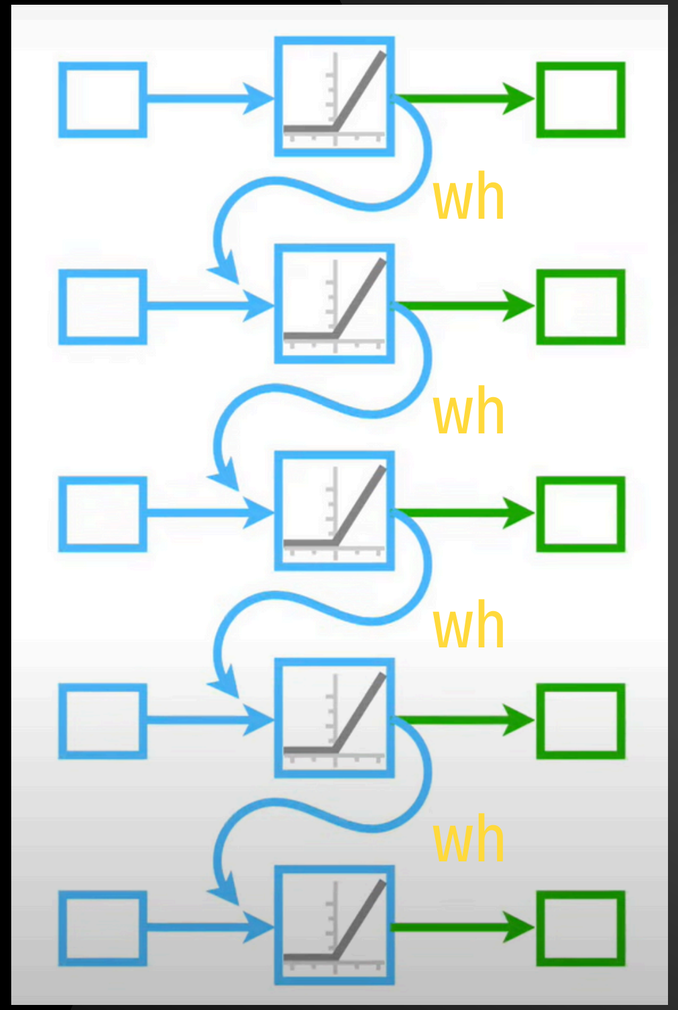
1. **Forward Pass**
2. **Error Calculation**
3. **Backpropagation Through Time (BPTT)**
4. **Gradient Descent**

$$h_t = f(W_h \cdot h_{t-1} + W_x \cdot x_t + b))$$

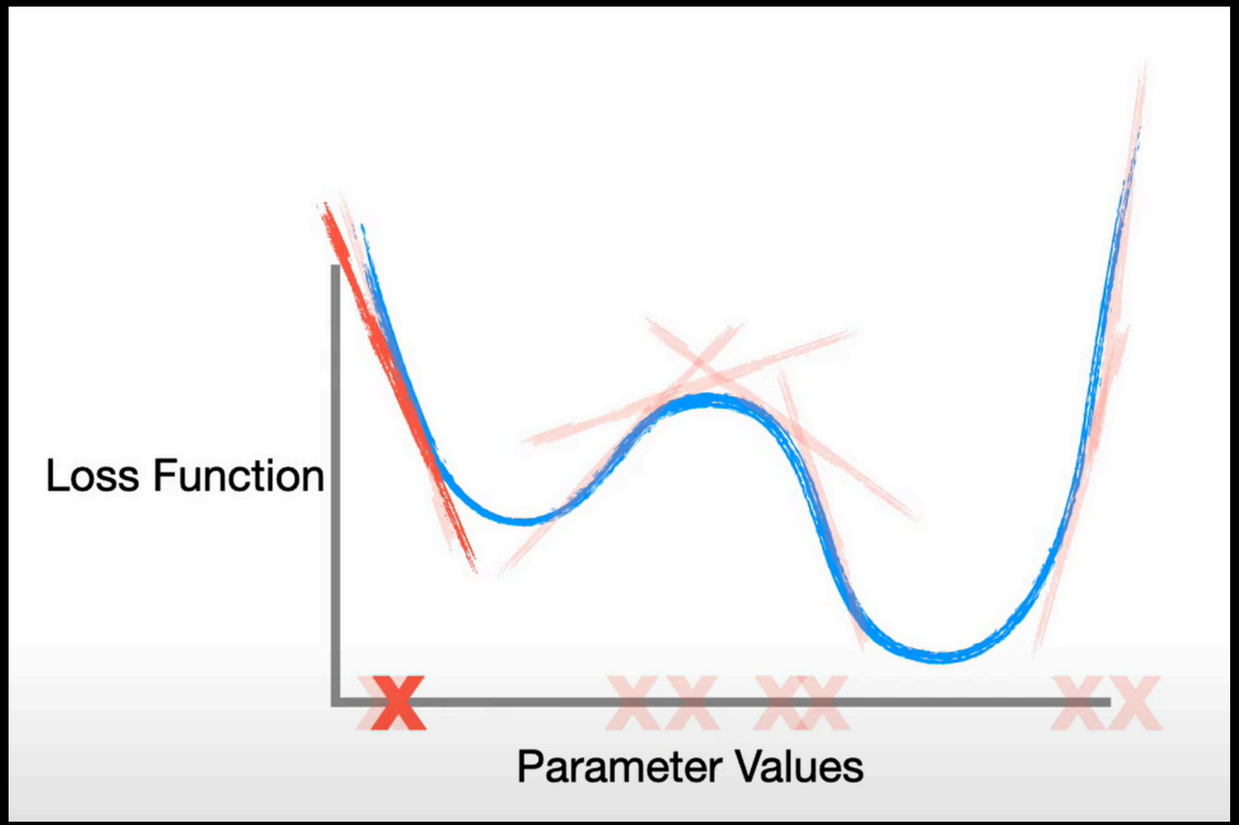
Challenges

<u>if the number of entries increases:</u>

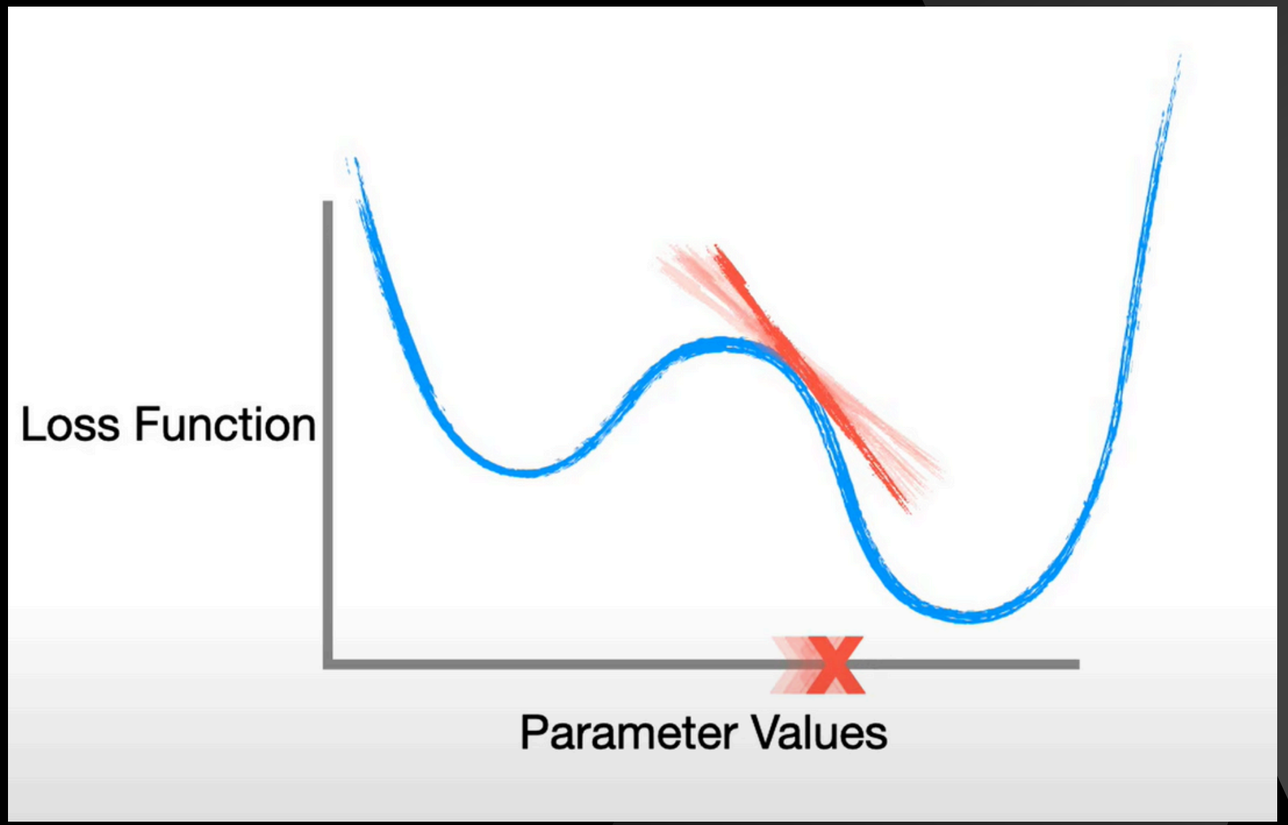
- Vanishing Gradient if wh < 1
- Exploding Gradient if wh > 1



Exploding Gradient



Vanishing Gradient



The need to improve

Variant

Mitigation of the vanishing and exploding Gradient issue

Better Memory

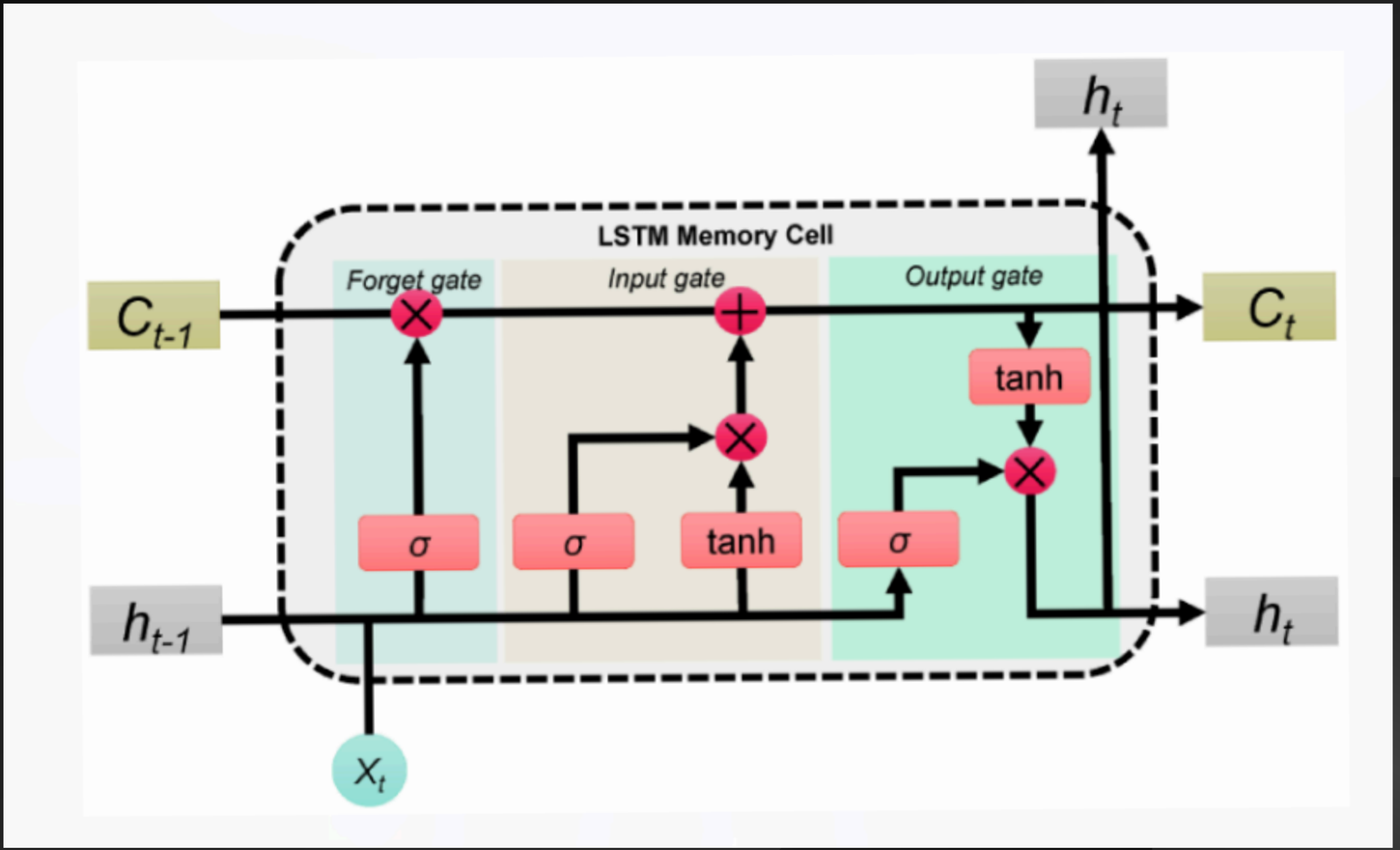
Long–term depencies maintained

Complexity

We can now address more complex problems

Long–Short Term Memory Neural Networks – LSTM

has memory through gates and cells



Forget Gate

decides what information to discard from the cell state.
adjusts forgetfulness by looking at h(t−1) and x(t)

Input Gate

determines what new information will be stored in the cell state
sigmoid decides what values to update
tanh provides potential candidates

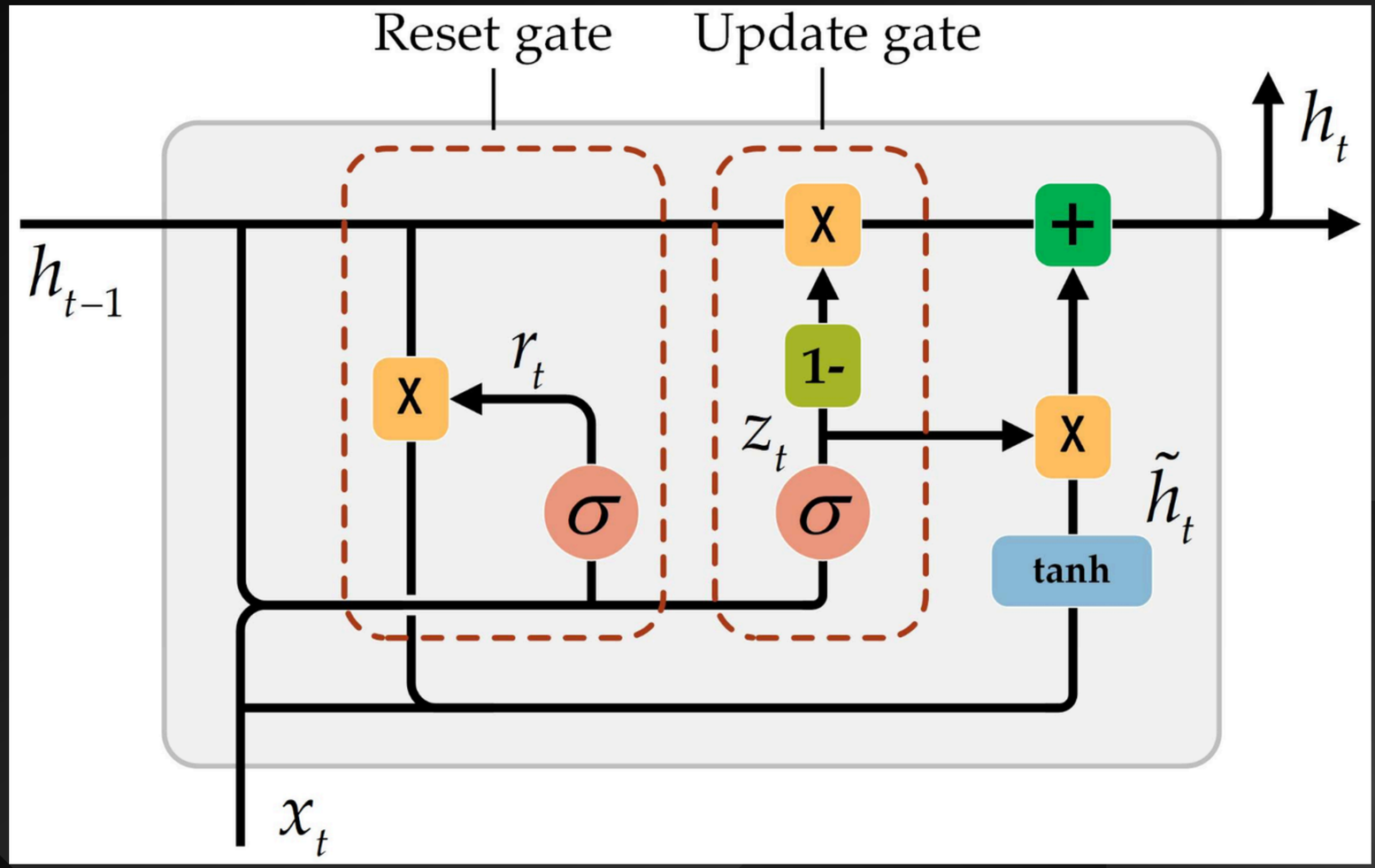
Output Gate

controls what part of the cell state will affect the output

Gated Recurrent Unit
GRU

GRU combines:

- the “forget” and “input” gates into an “update” gate
- the hidden and cell states h(t) and c(t) into one



Update Gate

determines how much of the old state is kept in the current state

Reset Gate

controls how much of the hidden state needs to be forgotten

RNN vs LSTM vs GRU

Feature	RNN	LSTM	GRU
Memory	Short-term only	Long-term memory via memory cells	Long-term via gates
Gates	No gates	Input, Forget, Output	Update, Reset
Complexity	Simple	Complex (3 gates)	Less Complex (2 gates)
Training Time	Fast	Slow	Faster than LSTM / Slower than RNN
Vanishing Gradient Problem	Yes	Solved using gates and memory cells	Solved using gates
Computational Cost	Low	High	Moderate
Suitability for Long-term Dependencies	Struggles	Well-suited	Effective yet simpler
Use Cases	Basic sequence data	Speech Recognition, Complex time-series analysis, machine translation	Similar to LSTM but for smaller datasets
Performance	Limited	High performance	Effective yet less than LSTM
Example Applications	Sentiment Analysis, basic forcasting	Text generation, speech synthesis, complex sequence learning	Machine translation, real-time speech recognition

Dataset Choice: Emotion analysis

emotion analysis (not sentiment)

larger dataset with a categorical label

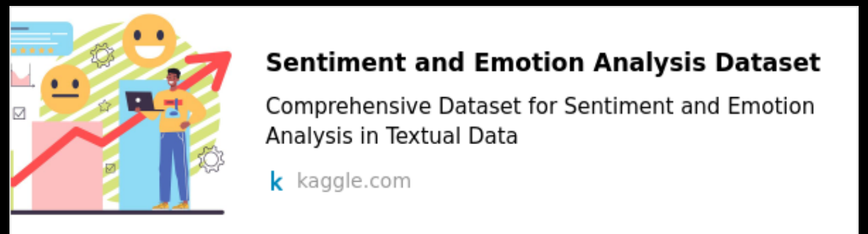
422746 entries

all non-null entries; we have some duplicates though

sentence feature, emotion label

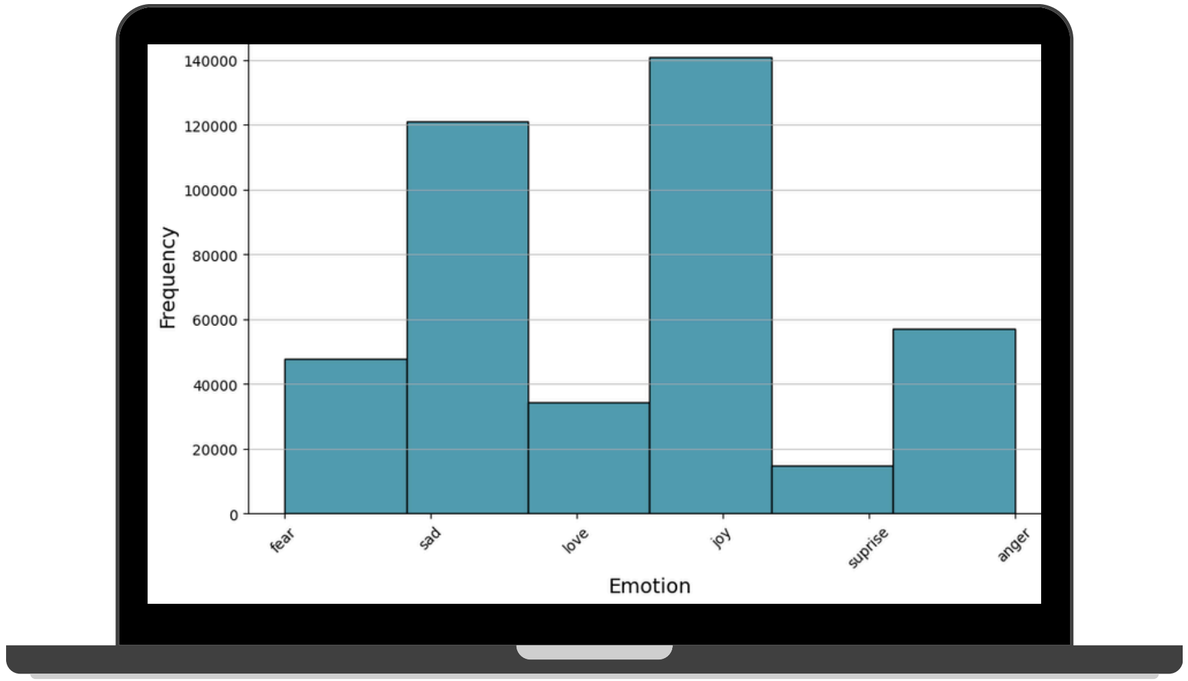
categorical label

id	sentence	emotion
422741	i begun to feel distressed for you	fear
422742	i left feeling annoyed and angry thinking that...	anger
422743	i were to ever get married i d have everything...	joy
422744	i feel reluctant in applying there because i w...	fear
422745	i just wanted to apologize to you because i fe...	anger



- usability: 10.0
- 2.23k downloads
- versatilility

Distribution of our classes



422746 total entries

Data Cleaning & Transformation

```python
def clean_text(text):

    # Convert words to lower case
    text = text.lower()

    # Use other preprocessing functions
    text = decontract_words(text)
    text = format_text_regex(text)

    # Tokenize each word
    text = remove_stopwords(text)
    text =  nltk.WordPunctTokenizer().tokenize(text)

    return text
```

```python
def lemmatized_words(text):
    lemm = nltk.stem.WordNetLemmatizer()
    df['lemmatized_text'] = list(map(lambda word:
                                     list(map(lemm.lemmatize, word)),
                                     df.Text_Cleaned))


lemmatized_words(df.Text_Cleaned)
```

```python
w2v_model = Word2Vec(df['lemmatized_text'], vector_size=300, window=5, min_count=3)
```

```python
df['emotion_label'] = df['emotion'].astype('category').cat.codes
y = df['emotion_label'].values  # we use integer labels
```

```python
df['vector'] = df['lemmatized_text'].apply(lambda x: text_to_vec(x, w2v_model))
```

Model Building

```python
class TextDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32).to(device)
        self.y = torch.tensor(y, dtype=torch.long).to(device)

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

train_dataset = TextDataset(X_train, y_train)
test_dataset = TextDataset(X_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=4, shuffle=True)
```

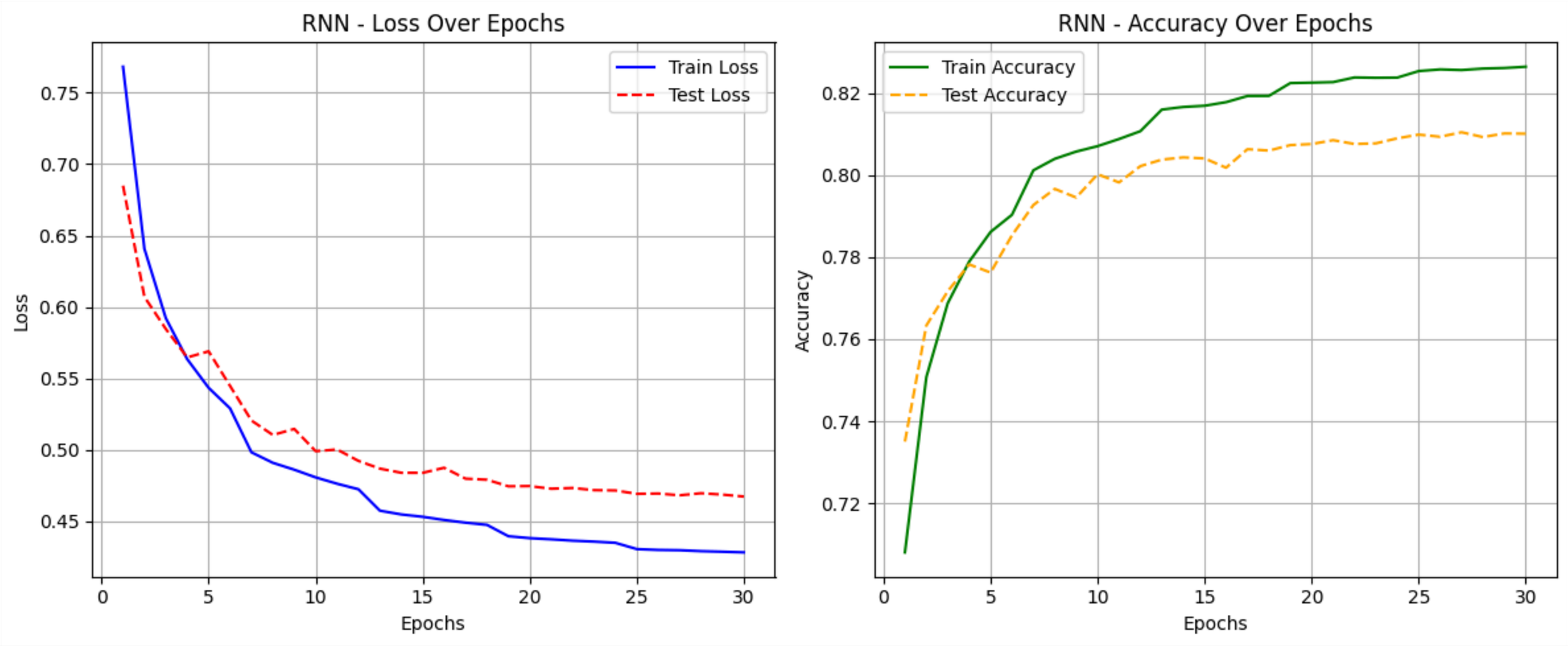
```python
# Base class for sequence models
class SequenceModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers, model_type):
        super(SequenceModel, self).__init__()
        self.model_type = model_type.lower()
        self.hidden_size = hidden_size
        self.num_layers = num_layers

        if self.model_type == "rnn":
            self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)
        elif self.model_type == "lstm":
            self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        elif self.model_type == "gru":
            self.gru = nn.GRU(input_size, hidden_size, num_layers, batch_first=True)
        else:
            raise ValueError("Invalid model_type. Choose from ['RNN', 'LSTM', 'GRU'].")

        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = x.unsqueeze(1)  # Adding a sequence length dimension (batch_size, seq_len=1, input_size)

        if self.model_type == "rnn":
            out, _ = self.rnn(x)
        elif self.model_type == "lstm":
            out, _ = self.lstm(x)
        elif self.model_type == "gru":
            out, _ = self.gru(x)

        out = self.fc(out[:, -1, :])  # Take the last output step
        return out
```

Evaluation Metrics

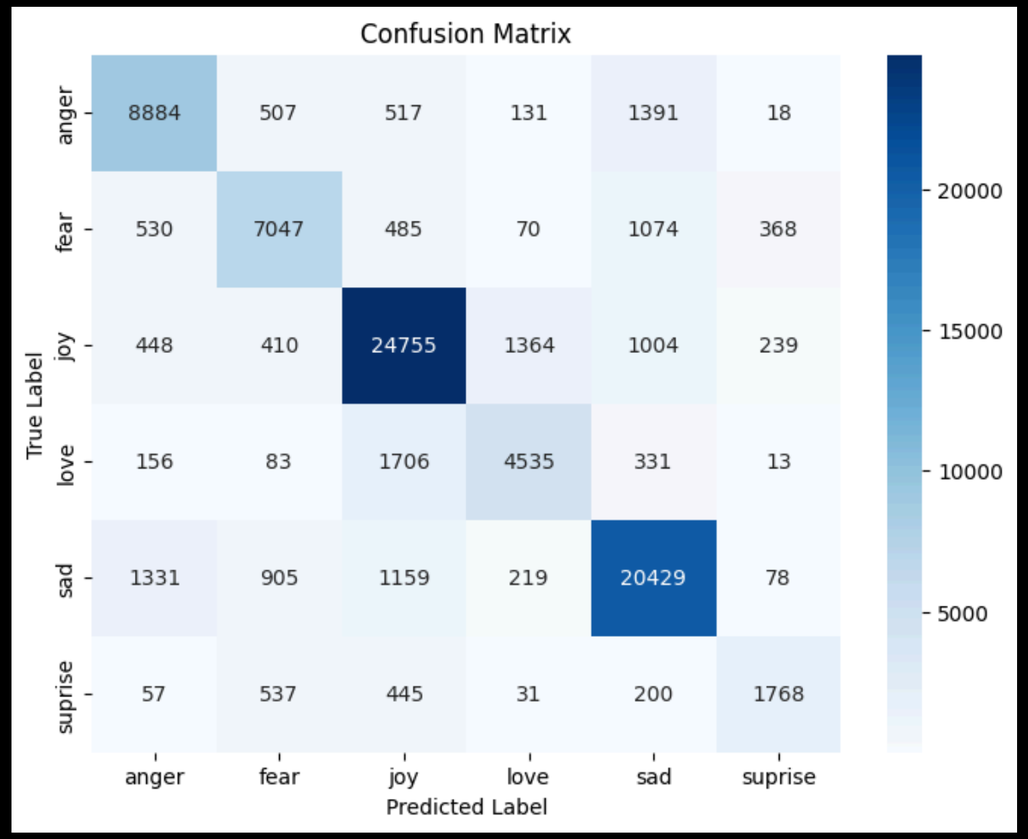
Since this is a multi-class dataset based on text and supervised classification, we need to evaluate the following metrics:

Metric	*Description*	*Use Case*
Accuracy	Percentage of correct predictions.	General performance metric.
Precision	Correct positive predictions out of all predicted positives.	Important for imbalanced datasets.
Recall	Correct positive predictions out of all actual positives.	Important when missing positives is costly.
F1-Score	Harmonic mean of Precision & Recall.	Best for imbalanced data.
Confusion Matrix	Shows misclassifications per class.	Analyzing model errors.

RNN, [300–64–64–6], lr=0.001, epochs=30



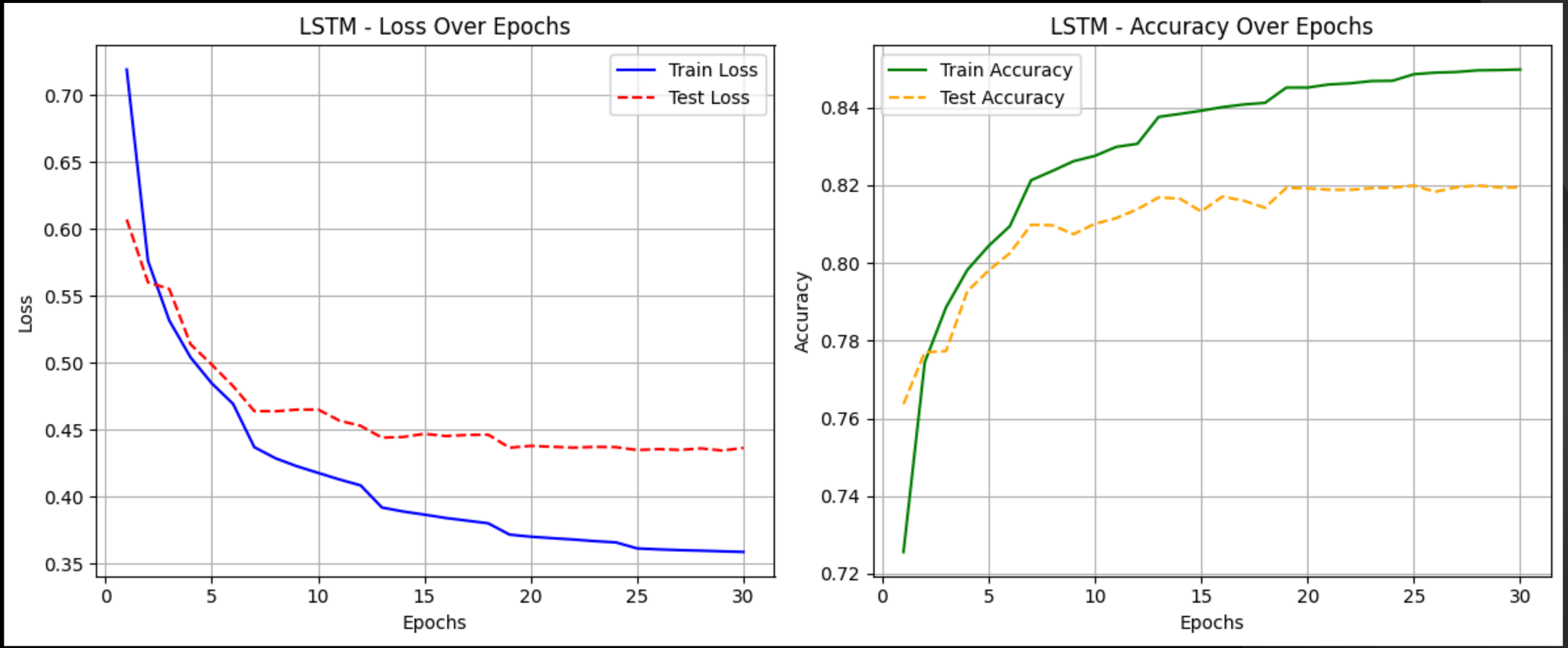
RNN, [300–64–64–6], lr=0.001, epochs=30



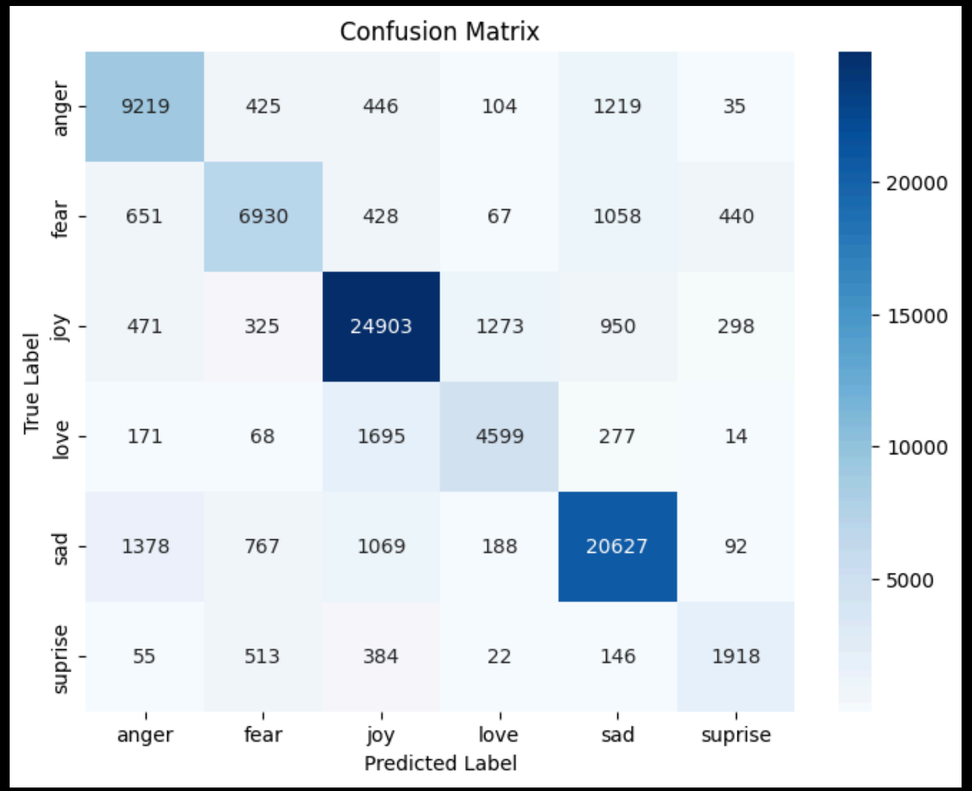
	precision	recall	f1-score	support
anger	0.78	0.78	0.78	11448
fear	0.74	0.74	0.74	9574
joy	0.85	0.88	0.86	28220
love	0.71	0.66	0.69	6824
sad	0.84	0.85	0.84	24121
suprise	0.71	0.58	0.64	3038
accuracy			0.81	83225
macro avg	0.77	0.75	0.76	83225
weighted avg	0.81	0.81	0.81	83225

Precision: 0.8083
Recall: 0.8101
F1 Score: 0.8088

LSTM, [300–64–64–6], lr=0.001, epochs=30



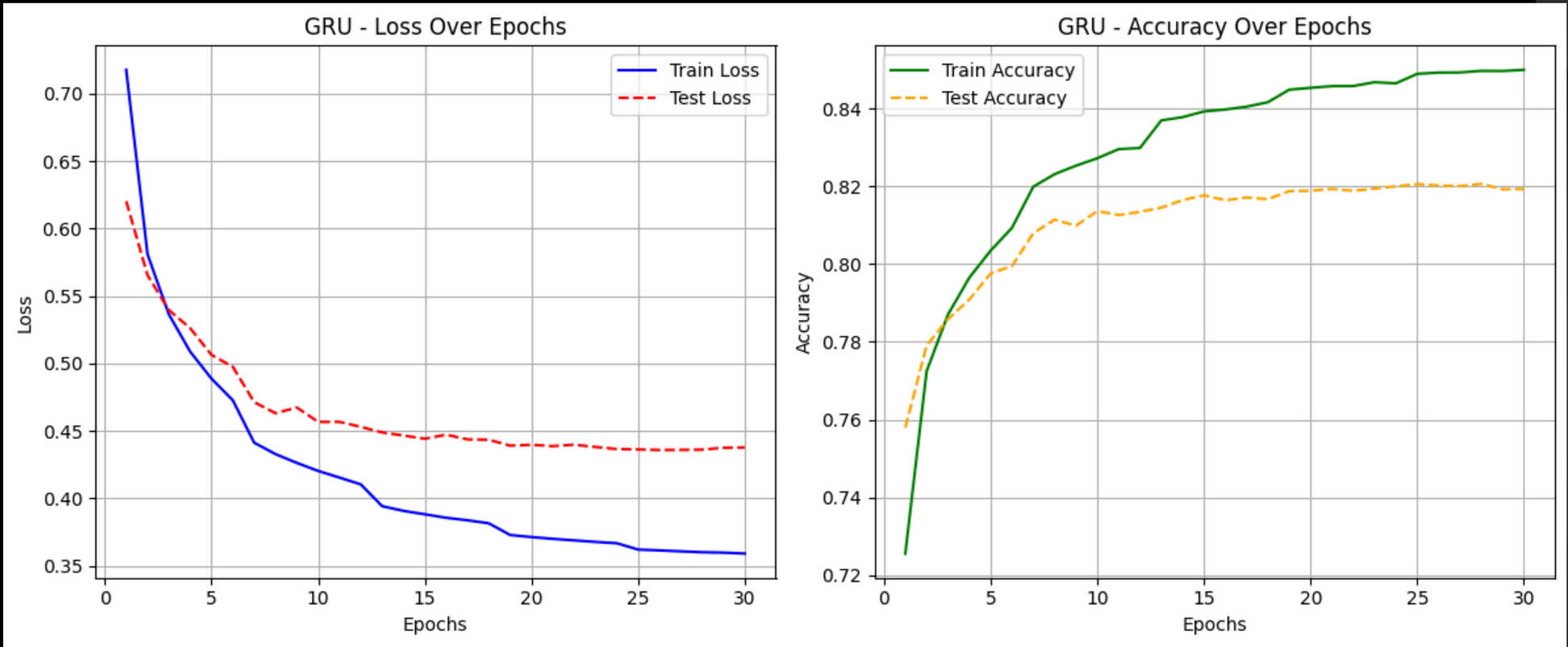
LSTM, [300–64–64–6], lr=0.001, epochs=30



	precision	recall	f1-score	support
anger	0.77	0.81	0.79	11448
fear	0.77	0.72	0.75	9574
joy	0.86	0.88	0.87	28220
love	0.74	0.67	0.70	6824
sad	0.85	0.86	0.85	24121
suprise	0.69	0.63	0.66	3038
accuracy			0.82	83225
macro avg	0.78	0.76	0.77	83225
weighted avg	0.82	0.82	0.82	83225

Precision: 0.8180
Recall: 0.8194
F1 Score: 0.8184

GRU, [300–64–64–6], lr=0.001, epochs=30

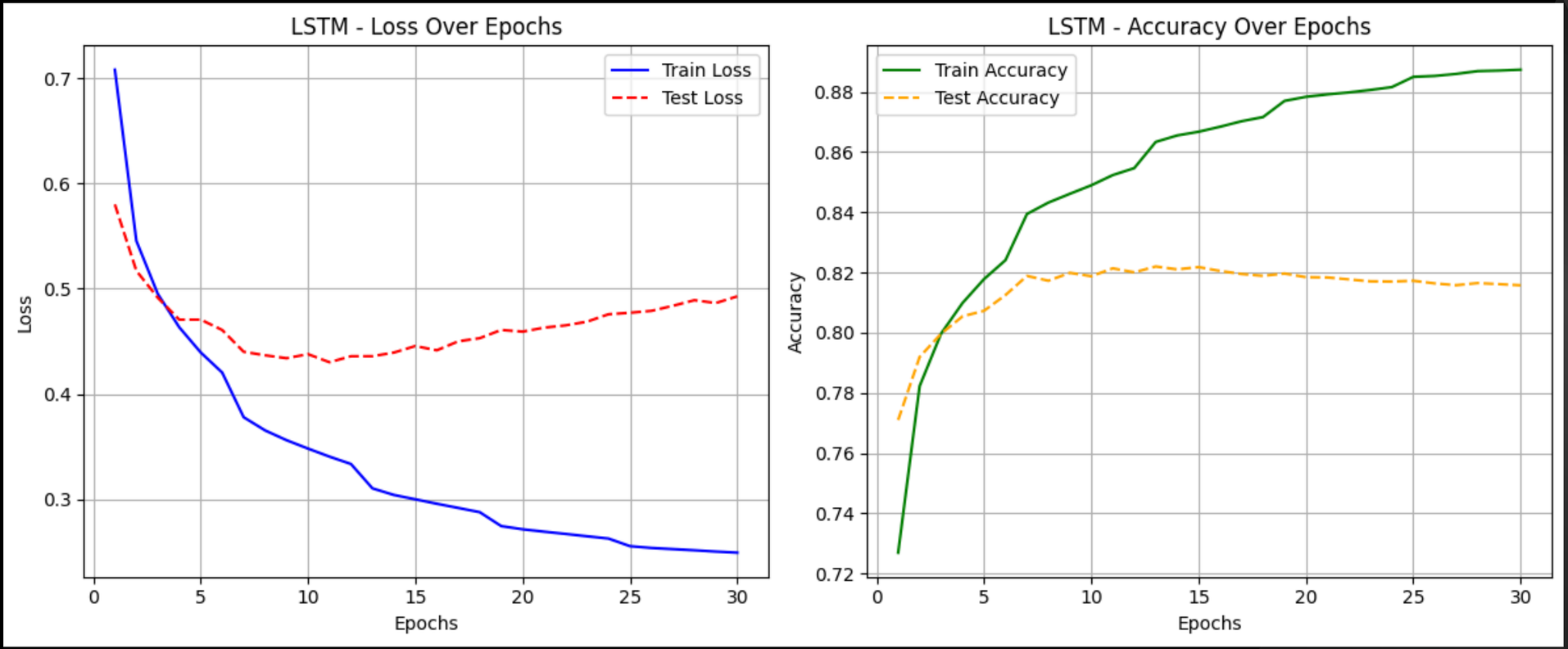


GRU, [300–64–64–6], lr=0.001, epochs=30

	precision	recall	f1-score	support
anger	0.77	0.81	0.79	11448
fear	0.76	0.74	0.75	9574
joy	0.86	0.89	0.87	28220
love	0.74	0.67	0.70	6824
sad	0.86	0.84	0.85	24121
suprise	0.67	0.65	0.66	3038
accuracy			0.82	83225
macro avg	0.78	0.77	0.77	83225
weighted avg	0.82	0.82	0.82	83225

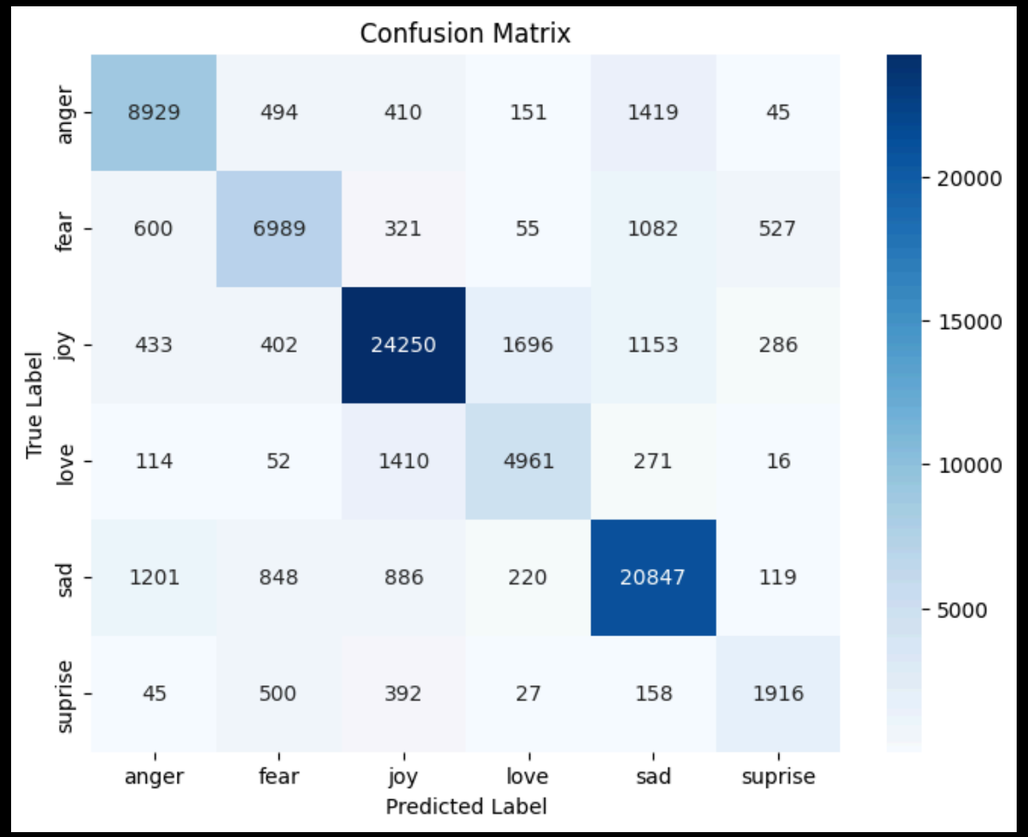
Precision: 0.8184
Recall: 0.8193
F1 Score: 0.8185

LSTM
[300–128–128–128–6], lr=0.001, epochs=30



Overfitting !!

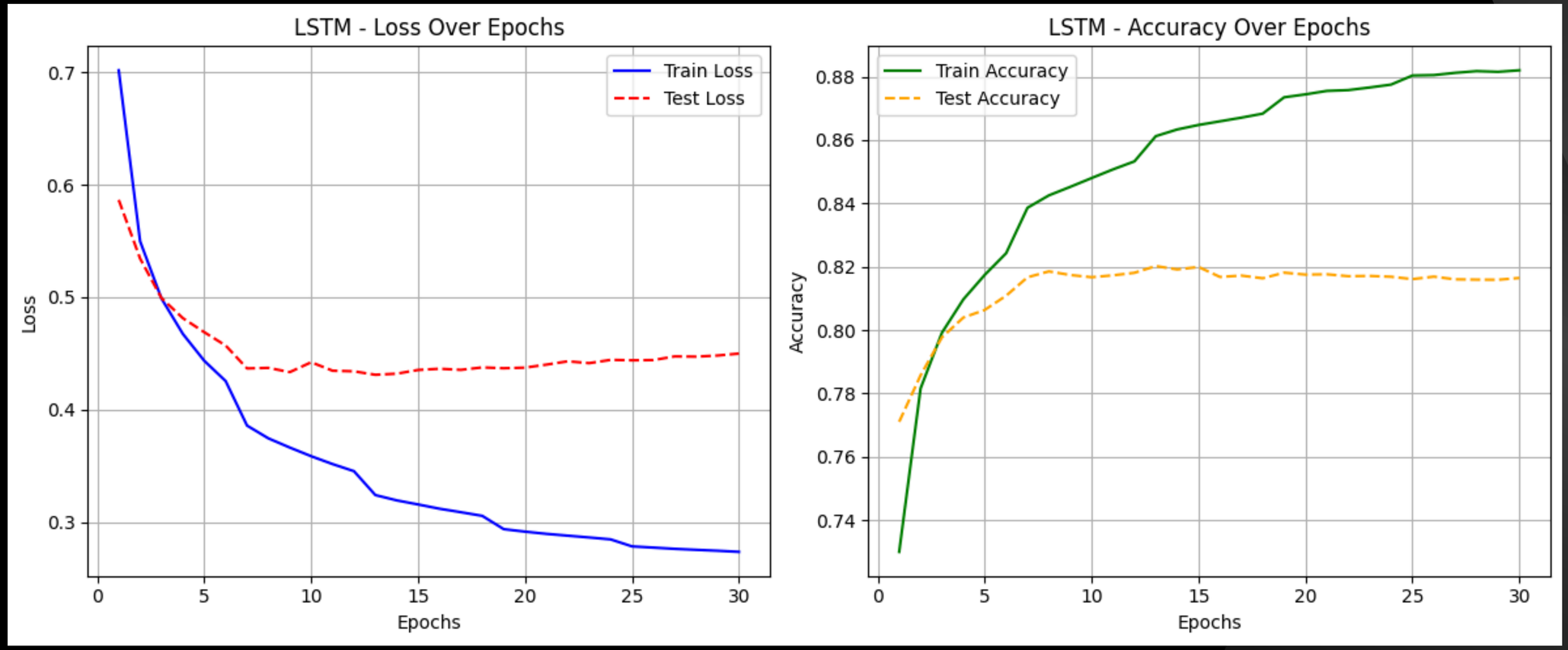
LSTM
[300–128–128–128–6], lr=0.001, epochs=30



	precision	recall	f1-score	support
anger	0.79	0.78	0.78	11448
fear	0.75	0.73	0.74	9574
joy	0.88	0.86	0.87	28220
love	0.70	0.73	0.71	6824
sad	0.84	0.86	0.85	24121
suprise	0.66	0.63	0.64	3038
accuracy			0.82	83225
macro avg	0.77	0.77	0.77	83225
weighted avg	0.82	0.82	0.82	83225

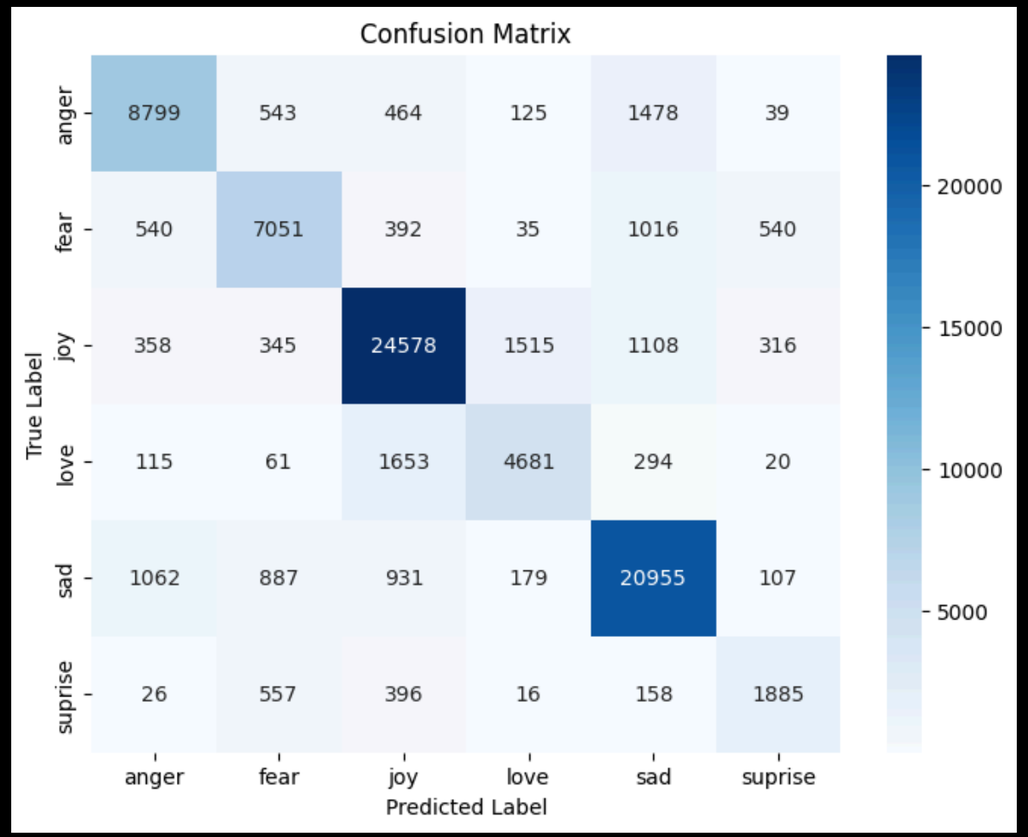
Precision: 0.8159
Recall: 0.8158
F1 Score: 0.8157

LSTM
[300–128–128–6], lr=0.001, epochs=30



Overfitting !!

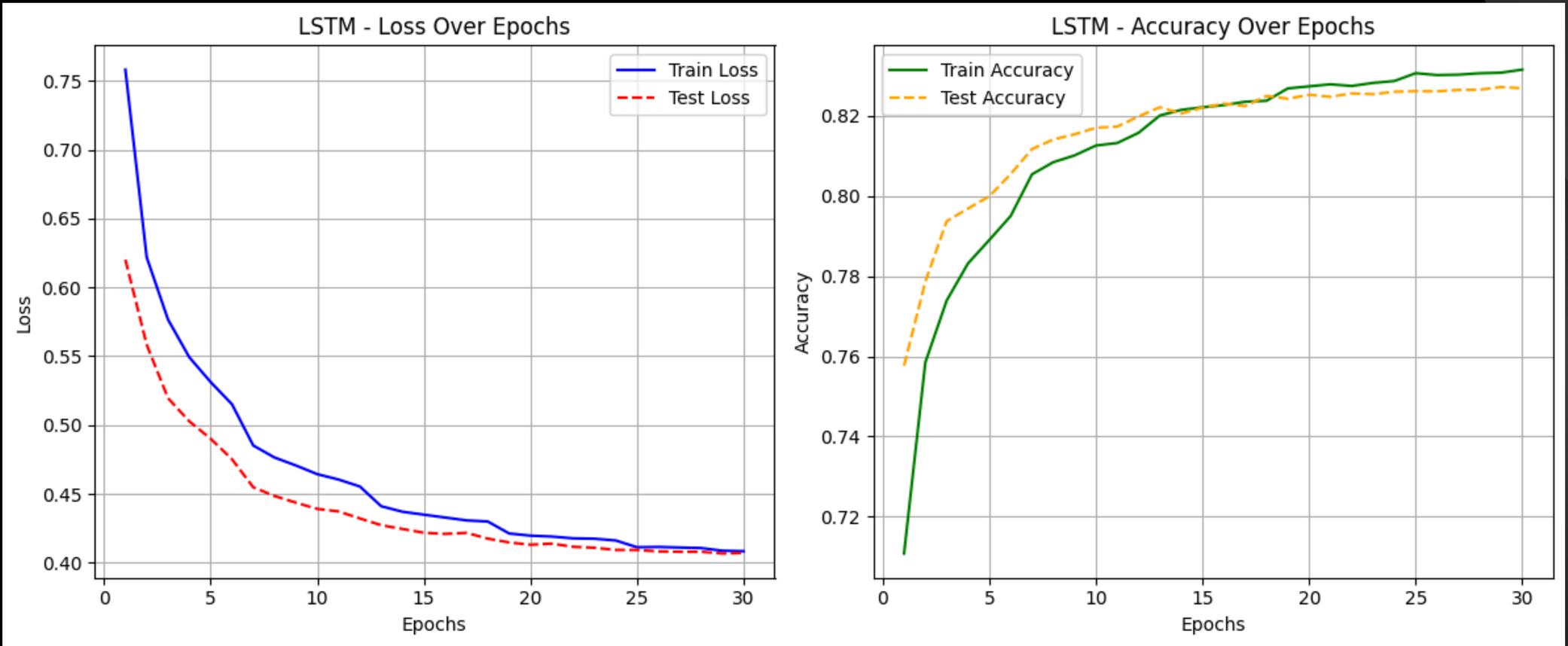
LSTM
[300–128–128–6], lr=0.001, epochs=30



	precision	recall	f1-score	support
anger	0.81	0.77	0.79	11448
fear	0.75	0.74	0.74	9574
joy	0.86	0.87	0.87	28220
love	0.71	0.69	0.70	6824
sad	0.84	0.87	0.85	24121
suprise	0.65	0.62	0.63	3038
accuracy			0.82	83225
macro avg	0.77	0.76	0.76	83225
weighted avg	0.82	0.82	0.82	83225

Precision: 0.8153
Recall: 0.8164
F1 Score: 0.8157

LSTM
[300–128–128–6], lr=0.001, epochs=30



Adding a 0.3 Dropout

LSTM
[300–128–128–6], lr=0.001, epochs=30

	precision	recall	f1-score	support
anger	0.81	0.79	0.80	11448
fear	0.76	0.75	0.76	9574
joy	0.87	0.88	0.88	28220
love	0.74	0.71	0.72	6824
sad	0.84	0.87	0.86	24121
suprise	0.72	0.62	0.67	3038
accuracy			0.83	83225
macro avg	0.79	0.77	0.78	83225
weighted avg	0.83	0.83	0.83	83225

Precision: 0.8254
Recall: 0.8268
F1 Score: 0.8259

Adding a 0.3 Dropout

Best Results

	precision	recall	f1-score	support
anger	0.81	0.79	0.80	11448
fear	0.76	0.75	0.76	9574
joy	0.87	0.88	0.88	28220
love	0.74	0.71	0.72	6824
sad	0.84	0.87	0.86	24121
suprise	0.72	0.62	0.67	3038
accuracy			0.83	83225
macro avg	0.79	0.77	0.78	83225
weighted avg	0.83	0.83	0.83	83225

Precision: 0.8254
Recall: 0.8268
F1 Score: 0.8259

Precision = 82.52%

F1–Score = 82.59%

Best Results

- word2vec embedding:
  - input_size = 300
  - min_count = 3
  - window = 3
- hyperparameters
  - model = “lstm”
  - hidden_size = 128
  - num_layers = 2
  - input_size = 300
  - output_size = 6
  - learning_rate = 0.001
  - epochs = 30
- Loss function: CrossEntropyLoss() from torch.nn

Precision = 82.52%

F1–Score = 82.59%

Click here to access the GitHub Repository

Challenges

- **embedding model parameter tuning:**
  - input_size = 300
  - min_count = 3
  - window = 3
- **hyperparameters tuning:**
  - model = “lstm”, “gru”, “rnn”
  - hidden_size = 128, 64, 96
  - num_layers = 2, 3
  - input_size = 300, ..., 50
  - output_size = 6
  - learning_rate = 0.001 (step)
  - epochs = 30
- **Loss function: CrossEntropyLoss() from torch.nn**

- **multiple combinations**
- **time consuming trial and error**

Next Steps

1. Use more advanced NLP models like BERT
2. Use better embedding models
3. Test and compare
4. Increase the dataset
5. Explore the dataset more

Thank you