Date of Submission July 7, 2025.

Text Classification Using Recurrent Neural Network

AMRIT KANDEL[1], (THA078BEI004), PRASISH TIMALSINA[2], (THA078BEI026)
[1]Department of Electronics and Computer Engineering, Thapathali Campus (e-mail: amrit.tha078bei004@tcioe.edu.np)
[2]Department of Electronics and Computer Engineering, Thapathali Campus (e-mail: prasish.tha078bei026@tcioe.edu.np)

ABSTRACT In this project, we worked on classifying movie reviews as positive or negative using recurrent neural networks. We created a small dataset of 100 manually labeled reviews and split it into training and testing sets. The text was processed using a basic tokenizer and converted into fixed-length sequences. Two models were used: a simple neural network and an LSTM model. After training both models, we found that the LSTM performed better because it understands word order in a sentence. This project helped us understand how deep learning can be used for text classification.

INDEX TERMS Recurrent Neural Networks (RNN), Text Classification, Long Short Term Memory (LSTM)

I. INTRODUCTION

A. RECURRENT NEURAL NETWORKS (RNN)

RNN is one of the popular neural networks that is commonly used to solve natural language processing tasks. This is a type of artificial neural network that can process sequential data, recognize patterns and predict the final output. This Neural Network is called Recurrent because it can repeatedly perform the same task or operation on a sequence of inputs. An RNN has an internal memory that allows it to remember or memorize the information of the input it received and this helps the system to gain context. RNNs feed information back into the network at each step.

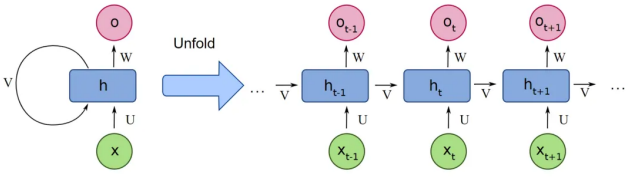

FIGURE 1. Block Diagram of Recurrent Neural Network

There are two key components in RNN. They are described below.

1) Recurrent Neuron

A recurrent neuron is the basic unit of an RNN which processes one element in a sequence at a time. In contrast to regular neurons, it has a memory to remember information from previous steps in the sequence. This is done by both the current input and previous output to produce the next output. Because of this, the RNN is able to learn sequentially occurring things like the order of words in a sentence.

2) RNN Unfolding

RNN unfolding is the method of unwrapping the network in time to show how it works step by step. The same neuron is constantly being reused, yet unfolding helps us see how the RNN cell processes each part of the sequence. Output from one neuron is used as the next neuron's input at each step. The identical repeated structure allows the RNN to process sequences of any length with the same weights at each step.

B. LONG SHORT TERM MEMORY (LSTM)

A Long short-term memory (LSTM) is a type of Recurrent Neural Network specially designed to prevent the neural network output for a given input from either decaying or exploding as it cycles through the feedback loops. The feedback loops allow recurrent networks to be better at pattern recognition than other neural networks. Memory of past input is critical for solving sequence learning tasks and Long short-term memory networks provide better performance compared to other RNN architectures.

LSTM architectures involves the memory cell which is controlled by three gates:

1) Forget Gate

The information that is no longer useful in the cell state is removed with the forget gate. Two inputs x_{t-1} (input at the particular time) and h_{t-1} (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias. The resultant is passed through an activation

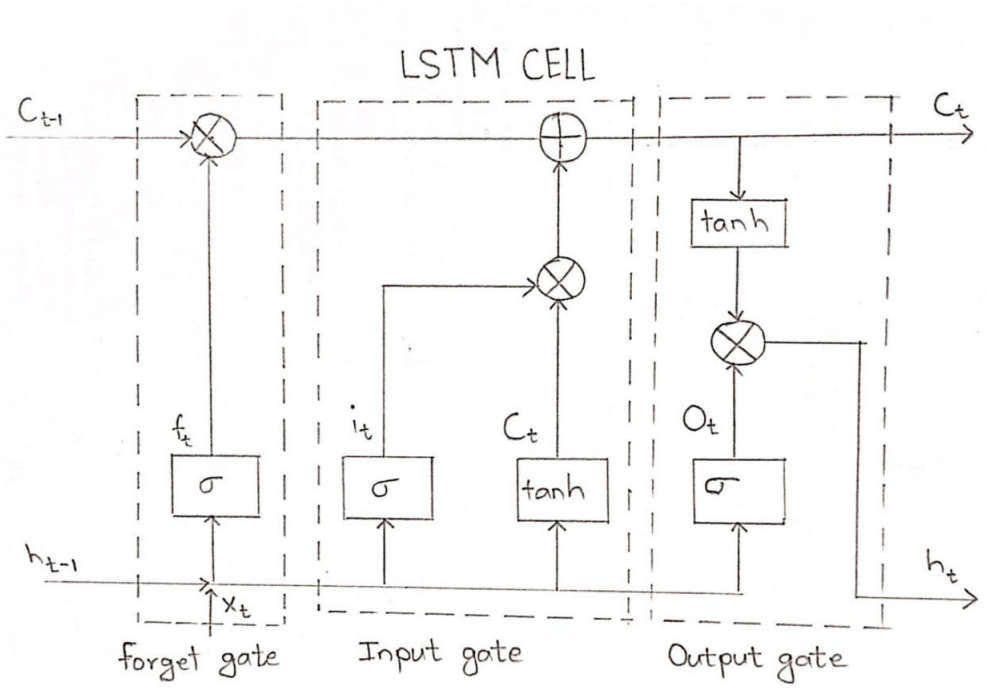

FIGURE 2. **Block Diagram of LSTM**

function which gives a binary output. If for a particular cell state the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use.

2) Input Gate

The addition of useful information to the cell state is done by the input gate. First the information is regulated using the sigmoid function and filter the values to be remembered similar to the forget gate using inputs $h_{t-1}$and x_t. Then, a vector is created using tanh function that gives an output from -1 to +1 which contains all the possible values from h_{t-1} and x_t. At last the values of the vector and the regulated values are multiplied to obtain the useful information.

3) Output Gate

The task of extracting useful information from the current cell state to be presented as output is done by the output gate. First, a vector is generated by applying tanh function on the cell. Then, the information is regulated using the sigmoid function and filter by the values to be remembered using inputs h_{t-1} and x_t. At last the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell.

II. DATASET DESCRIPTION

In this project, we manually created a small dataset consisting of 100 short movie reviews, each labeled to indicate sentiment: label 0 for negative reviews and label 1 for positive reviews. The dataset contains simple natural language sentences expressing opinions about movies. To train and evaluate the model, the dataset was split into 80% for training set and 20% for testing set. A vocabulary was then built from all the data (both training and testing) to ensure that every word encountered during inference would be recognized by the model. This setup allows for a clear demonstration of text classification using a basic neural network on a manageable dataset.

III. EXPERIMENTATION MODEL

In this project, we used two models for text classification: a simple ANN and an LSTM-based model. Before feeding the data into these models, each review datum was processed using a tokenizer, which converted text to lowercase, removed punctuation, and split the text into individual words. A vocabulary was built from all the reviews, assigning a unique index to each word.

Each review was then encoded as a fixed-length sequence of word indices. If a review was shorter than the defined maximum length, it was padded; if longer, it was truncated. These sequences served as the input to both models.

The first model, the ANN, consists of an embedding layer followed by mean pooling over the sequence, then two fully connected layers with a ReLU activation in between. The second model uses an LSTM layer after the embedding to better capture word order and context within the review. The output from the LSTM was passed to a fully connected layer for final classification.

IV. RESULTS

When both models were trained and evaluated, following results were observed.

A. FOR ANN MODEL

```
Epoch [1/25]     Loss: 0.7142    Accuracy: 0.5250
Epoch [2/25]     Loss: 0.7096    Accuracy: 0.5250
Epoch [3/25]     Loss: 0.7056    Accuracy: 0.5250
Epoch [4/25]     Loss: 0.7009    Accuracy: 0.5250
Epoch [5/25]     Loss: 0.6978    Accuracy: 0.5250
Epoch [6/25]     Loss: 0.6954    Accuracy: 0.5250
Epoch [7/25]     Loss: 0.6905    Accuracy: 0.5250
Epoch [8/25]     Loss: 0.6879    Accuracy: 0.5250
Epoch [9/25]     Loss: 0.6834    Accuracy: 0.5250
Epoch [10/25]    Loss: 0.6814    Accuracy: 0.5250
Epoch [11/25]    Loss: 0.6763    Accuracy: 0.5500
Epoch [12/25]    Loss: 0.6719    Accuracy: 0.5875
Epoch [13/25]    Loss: 0.6667    Accuracy: 0.6250
Epoch [14/25]    Loss: 0.6633    Accuracy: 0.6625
Epoch [15/25]    Loss: 0.6562    Accuracy: 0.7000
Epoch [16/25]    Loss: 0.6501    Accuracy: 0.7250
Epoch [17/25]    Loss: 0.6443    Accuracy: 0.7500
Epoch [18/25]    Loss: 0.6364    Accuracy: 0.7375
Epoch [19/25]    Loss: 0.6284    Accuracy: 0.7375
Epoch [20/25]    Loss: 0.6189    Accuracy: 0.7625
Epoch [21/25]    Loss: 0.6098    Accuracy: 0.7750
Epoch [22/25]    Loss: 0.5995    Accuracy: 0.7625
Epoch [23/25]    Loss: 0.5890    Accuracy: 0.7625
Epoch [24/25]    Loss: 0.5779    Accuracy: 0.7625
Epoch [25/25]    Loss: 0.5659    Accuracy: 0.7625
...
    accuracy                           0.35        20
   macro avg       0.37      0.38      0.35        20
weighted avg       0.38      0.35      0.34        20
```

FIGURE 3. ANN Training Result

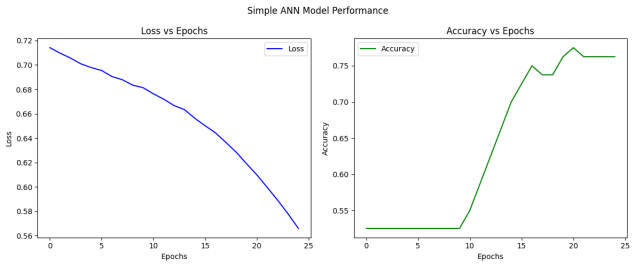

FIGURE 4. ANN Model Performance

B. FOR LSTM MODEL

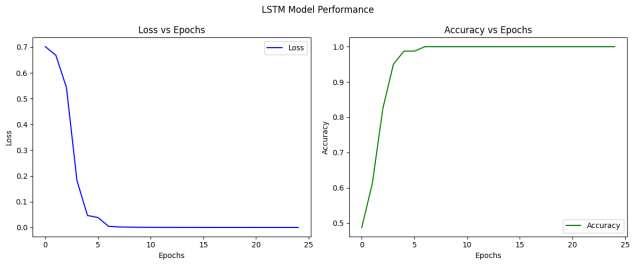

FIGURE 5. LSTM Model Performance

```
Epoch [1/25]     Loss: 0.7016    Accuracy: 0.4875
Epoch [2/25]     Loss: 0.6681    Accuracy: 0.6125
Epoch [3/25]     Loss: 0.5434    Accuracy: 0.8250
Epoch [4/25]     Loss: 0.1820    Accuracy: 0.9500
Epoch [5/25]     Loss: 0.0463    Accuracy: 0.9875
Epoch [6/25]     Loss: 0.0384    Accuracy: 0.9875
Epoch [7/25]     Loss: 0.0044    Accuracy: 1.0000
Epoch [8/25]     Loss: 0.0019    Accuracy: 1.0000
Epoch [9/25]     Loss: 0.0012    Accuracy: 1.0000
Epoch [10/25]    Loss: 0.0007    Accuracy: 1.0000
Epoch [11/25]    Loss: 0.0005    Accuracy: 1.0000
Epoch [12/25]    Loss: 0.0003    Accuracy: 1.0000
Epoch [13/25]    Loss: 0.0003    Accuracy: 1.0000
Epoch [14/25]    Loss: 0.0002    Accuracy: 1.0000
Epoch [15/25]    Loss: 0.0002    Accuracy: 1.0000
Epoch [16/25]    Loss: 0.0001    Accuracy: 1.0000
Epoch [17/25]    Loss: 0.0001    Accuracy: 1.0000
Epoch [18/25]    Loss: 0.0001    Accuracy: 1.0000
Epoch [19/25]    Loss: 0.0001    Accuracy: 1.0000
Epoch [20/25]    Loss: 0.0001    Accuracy: 1.0000
Epoch [21/25]    Loss: 0.0001    Accuracy: 1.0000
Epoch [22/25]    Loss: 0.0001    Accuracy: 1.0000
Epoch [23/25]    Loss: 0.0000    Accuracy: 1.0000
Epoch [24/25]    Loss: 0.0000    Accuracy: 1.0000
Epoch [25/25]    Loss: 0.0000    Accuracy: 1.0000
...
    accuracy                           0.70        20
   macro avg       0.73      0.73      0.70        20
weighted avg       0.76      0.70      0.70        20
```

FIGURE 6. LSTM Training Result

V. DISCUSSION

The results show that the LSTM model performed better than the simple feedforward neural network.LSTMs are designed to handle sequential data and can remember the order of words, which plays a key role in understanding sentiment in text. ANN uses only mean pooling, which loses word order information and treats all words equally, regardless of their position in the sentence.

However, due to the small dataset size, both models were limited in their learning capability. With only 100 samples, overfitting was observed as there was huge difference in test accuracy and training accuracy. Still, the experiment demonstrates the value of using a sequence-aware model like LSTM for natural language tasks.

VI. CONCLUSION

In this project, we compared two models for classification of text: a simple artificial neural network and an LSTM. Both models were implemented and evaluated on a small, manually labeled movie review dataset. While the ANN provided a simple and fast approach, the LSTM delivered better results by leveraging word order and sequence information. The experiment highlights the importance of model choice in text classification and suggests that recurrent architectures like LSTM are more effective when working with sequential data. For further improvement, using a larger and more diverse dataset would help train more accurate and robust models.

APPENDIX A
SOURCE CODE

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
import random
from sklearn.model_selection import
    train_test_split

## dataset
data = [
    # Positive
    ("I love this movie, it was fantastic!", 1),
    ("Amazing plot and great performances!", 1),
    ("What a wonderful experience!", 1),
    ("Brilliant direction and strong cast.", 1),
    ("Absolutely loved it!", 1),
    ("A masterpiece of modern cinema.", 1),
    ("The visuals and story were breathtaking.",
        1),
    ("Exceptional acting and brilliant screenplay.
        ", 1),
    ("Truly inspiring and well-crafted.", 1),
    ("An emotional rollercoaster with a happy end.
        ", 1),
    ("The movie exceeded my expectations.", 1),
    ("I enjoyed every minute of it.", 1),
    ("Heartwarming and deeply moving.", 1),
    ("A compelling story with great characters.",
        1),
    ("The actors delivered fantastic performances.
        ", 1),
    ("An unforgettable experience.", 1),
    ("The direction was top-notch.", 1),
    ("Highly entertaining and beautifully shot.",
        1),
    ("Best movie I've seen this year!", 1),
    ("Wonderful execution and pacing.", 1),
    ("Left me speechless, in a good way.", 1),
    ("Incredible storytelling and visuals.", 1),
    ("Uplifting and inspiring movie.", 1),
    ("A joy to watch from start to finish.", 1),
    ("Loved the soundtrack and cinematography.",
        1),
    ("Totally worth watching again!", 1),
    ("The dialogue was witty and smart.", 1),
    ("Very touching and well-acted.", 1),
    ("The characters felt very real and human.",
        1),
    ("A fresh and original take on a familiar
        genre.", 1),
    ("The chemistry between the leads was amazing.
        ", 1),
    ("Impressive production quality and plot depth
        .", 1),
    ("Simply stunning and emotionally satisfying."
        , 1),
    ("Loved the twists and turns.", 1),
    ("This film deserves all the praise.", 1),
    ("I was completely immersed.", 1),
    ("It was okay, not the best but enjoyable.",
        1),
    ("Pleasantly surprised by how good it was.",
        1),
    ("A solid movie night pick.", 1),
    ("Engaging and thoughtful throughout.", 1),
    ("I smiled through most of the film.", 1),
    ("Definitely recommend this movie.", 1),
    ("A positive and feel-good story.", 1),
    ("An all-time favorite!", 1),
    ("Five stars from me!", 1),
    ("A nice blend of humor and emotion.", 1),
    ("Very entertaining.", 1),
    ("One of the better films this year.", 1),
    ("Charming and full of heart.", 1),
    ("Didn't expect to love it this much.", 1),

    # Negative
    ("Absolutely terrible. Waste of time.", 0),
    ("Worst acting I have ever seen.", 0),
    ("I didn't like it at all.", 0),
    ("The film was boring and predictable.", 0),
    ("Horrible. I walked out halfway.", 0),
    ("A complete disaster from start to finish.",
        0),
    ("Terribly written and poorly executed.", 0),
    ("The plot made no sense.", 0),
    ("A mess of clich s and bad dialogue.", 0),
    ("Not worth the hype.", 0),
    ("It was painful to sit through.", 0),
    ("The pacing was way off.", 0),
    ("Characters were flat and uninteresting.", 0)
        ,
    ("Too many plot holes to count.", 0),
    ("Acting was wooden and emotionless.", 0),
    ("An insult to the genre.", 0),
    ("Predictable and lazy writing.", 0),
    ("Music was distracting and out of place.", 0)
        ,
    ("Uninspired and forgettable.", 0),
    ("Dialogue felt forced and unnatural.", 0),
    ("It dragged on forever.", 0),
    ("C o u l d n t connect with any character.", 0),
    ("The trailer was better than the movie.", 0),
    ("Disappointing and dull.", 0),
    ("Felt like a waste of money.", 0),
    ("The jokes were cringe-worthy.", 0),
    ("It lacked depth and originality.", 0),
    ("Too predictable and not funny.", 0),
    ("Very slow and uninteresting plot.", 0),
    ("Nothing happened for an hour.", 0),
    ("Really bad camera work and sound.", 0),
    ("The ending was abrupt and made no sense.",
        0),
    ("Failed to deliver any emotion.", 0),
    ("Overacted and overhyped.", 0),
    ("Not even good for background noise.", 0),
    ("Terribly boring.", 0),
    ("I regret watching it.", 0),
    ("Very disappointing.", 0),
    ("One of the worst films I've seen.", 0),
    ("Unbelievably bad.", 0),
    ("A total flop.", 0),
    ("The cast seemed confused.", 0),
    ("The story d i d n t go anywhere.", 0),
    ("I almost fell asleep.", 0),
    ("Nothing redeeming about it.", 0),
    ("It tried too hard to be deep.", 0),
    ("Just plain bad.", 0),
    ("I wanted to leave early.", 0),
    ("Bad acting, worse writing.", 0),
    ("Complete waste of time.", 0),
]

random.shuffle(data)
## define a tokenizer
def tokenizer(text):
    return text.lower().replace(',', '').replace('
        .', '').split()

# Create a vocabulary of all data
vocab = {"":0}
for text, _ in data:
    for token in tokenizer(text):
        if token not in vocab:
            vocab[token] = len(vocab)
```

```
len(vocab)
print(vocab)

## To encode for each line of text
#  we assume each line of text have maximum 10
    words

def encode_text(text):
    max_len = 10
    index_list = []
    tokens = tokenizer(text)
    for token in tokens:
        index_list.append(vocab.get(token, vocab["
            "]))

    index_list += [vocab[""]] * (max_len-len(
        index_list))
    index_list = index_list[:max_len]
    return torch.tensor(index_list)

print(encode_text("Best movie."))
print(encode_text("This movie is the worst movie I
     have ever watched. I regret watching it"))
# print(vocab.get('my'))

# Prepare the datset
class CustomDataset(Dataset):
    def __init__(self, data, max_len=10):
        self.data = data
        self.max_len = max_len

    def __getitem__(self, index):
        # return self.data[index]
        text, label = self.data[index]
        encoded_text = encode_text(text)  # Encode
            the text
        return encoded_text, torch.tensor(label)
            # Return tensors


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


train_data = data[:int(len(data)*0.8)]
test_data = data[int(len(data)*0.8):]

train_loader = DataLoader(CustomDataset(train_data
    ), batch_size = 2, shuffle=True)
test_loader = DataLoader(CustomDataset(test_data),
     batch_size = 2, shuffle=False)

## Creating simple ANN model
class SimpleANN(nn.Module):
    def __init__(self, vocab_size, embedding_dim
        =10):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size,
            embedding_dim)
        self.fc1 = nn.Linear(embedding_dim, 8)
        self.fc2 = nn.Linear(8,2)

    def forward(self, x):
        x = self.embedding(x)
        x = self.fc1(x.mean(dim=1))
        x = torch.relu(x)
        x = self.fc2(x)
        return x

## Using LSTM

class LSTM(nn.Module):
    def __init__(self, vocab_size, max_len):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size,
            100)
        self.lstm = nn.LSTM(100, 100, batch_first=
            True)
        self.fc1 = nn.Linear(100 * max_len, 50)
        self.fc2 = nn.Linear(50, 2)

    def forward(self, x):
        x = self.embedding(x)
        x, y = self.lstm(x)
        # print(x,y)
        x = torch.relu(x)
        x = x.reshape(x.size(0), -1)
        x = torch.relu(self.fc1(x))
        return self.fc2(x)



# function to train the model
from sklearn.metrics import accuracy_score,
    classification_report

def train_model(model, train_loader, criterion,
    optimizer, num_epochs=5):
    model.train()
    loss_history, accuracy_history = [], []
    for epoch in range(num_epochs):
        total_loss = 0
        all_preds = []
        all_labels = []
        for texts, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(texts)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            total_loss += loss.item()
            prediction = torch.argmax(outputs, dim
                =1)
            all_preds.extend(prediction.tolist())
            all_labels.extend(labels.tolist())


        avg_loss = total_loss / len(train_loader)
        accuracy = accuracy_score(all_labels,
            all_preds)
        loss_history.append(avg_loss)
        accuracy_history.append(accuracy)

        print(f'Epoch [{epoch+1}/{num_epochs}] \t
            Loss: {avg_loss:.4f} \t Accuracy: {
            accuracy:.4f}')

    return loss_history, accuracy_history

## function to evaluate the model
def evaluate_model(model, test_loader):
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for texts, labels in test_loader:
            outputs = model(texts)
            prediction = torch.argmax(outputs, dim
                =1)
            all_preds.extend(prediction.tolist())
            all_labels.extend(labels.tolist())

    accuracy = accuracy_score(all_labels,
        all_preds)
    print(f'Test Accuracy: {accuracy:.4f}')
```

```
    print(classification_report(all_labels,
        all_preds, target_names=["Negative", "
        Positive"]))
    return accuracy


## function to plot the results
import matplotlib.pyplot as plt
def plot_results(losses, accuracies, title="Model
    Performance"):
    plt.figure(figsize=(12,5))

    # Plot loss
    plt.subplot(1,2,1)
    plt.plot(losses, label="Loss", color='blue')
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.title("Loss vs Epochs")
    plt.legend()

    # Plot accuracy
    plt.subplot(1,2,2)
    plt.plot(accuracies, label="Accuracy", color='
        green')
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.title("Accuracy vs Epochs")
    plt.legend()

    plt.suptitle(title)
    plt.tight_layout()
    plt.show()



## for simple ANN model
ann_model = SimpleANN(len(vocab))
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(ann_model.parameters
    (), lr=0.001)
losses, accuracies = train_model(ann_model,
    train_loader, criterion, optimizer, num_epochs
    =25)
evaluate_model(ann_model, test_loader)
plot_results(losses, accuracies, title="Simple ANN
     Model Performance")

## For LSTM model
lstm_model = LSTM(len(vocab), max_len=10)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(lstm_model.parameters
    (), lr=0.001)
losses, accuracies = train_model(lstm_model,
    train_loader, criterion, optimizer, num_epochs
    =25)
evaluate_model(lstm_model, test_loader)
plot_results(losses, accuracies, title="LSTM Model
     Performance")
```

• • •