Natural Language Processing- Assignment - 4

Name: Adithi Shinde

Enrollment No: 2203A54032

Batch - 40

1.Load data fromkeras.datasets and perform following computational analysis:- [CO2]

(a) Preprocessing of the Data

```python
# Import necessary libraries
import pandas as pd
import numpy as np
import tensorflow as tf

# Load the dataset (update the filename if needed)
data = pd.read_csv('IMDB Dataset.csv')  # Replace with your actual file name if different

# Display the first few rows of the dataset
print(data.head())

# Check for missing values
print(data.isnull().sum())

# Tokenize and pad the text sequences
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=10000)  # Top 10,000 words
tokenizer.fit_on_texts(data['review'])  # Fit tokenizer on text data

# Convert texts to sequences
X = tokenizer.texts_to_sequences(data['review'])

# Pad sequences to ensure uniform input size
max_length = 250  # Define max length for padding
X = tf.keras.preprocessing.sequence.pad_sequences(X, maxlen=max_length)

# Convert sentiment labels to binary (0 for negative, 1 for positive)
y = np.where(data['sentiment'] == 'positive', 1, 0)

# Check the shapes of the resulting arrays
print(f'Shape of X: {X.shape}')
print(f'Shape of y: {y.shape}')
```

```
                                              review sentiment
0  One of the other reviewers has mentioned that ...  positive
1  A wonderful little production. <br /><br />The...  positive
2  I thought this was a wonderful way to spend ti...  positive
3  Basically there's a family where a little boy ...  negative
4  Petter Mattei's "Love in the Time of Money" is...  positive
review       0
sentiment    0
dtype: int64
Shape of X: (50000, 250)
Shape of y: (50000,)
```

(b)Divide data into training and testing data set :

```python
from sklearn.model_selection import train_test_split

# Assuming X (features) and y (labels) are already defined from preprocessing
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Check the shapes of the resulting datasets
print(f'Shape of X_train: {X_train.shape}')
print(f'Shape of X_test: {X_test.shape}')
print(f'Shape of y_train: {y_train.shape}')
print(f'Shape of y_test: {y_test.shape}')
```

```
Shape of X_train: (40000, 250)
Shape of X_test: (10000, 250)
Shape of y_train: (40000,)
Shape of y_test: (10000,)
```

(c)Build the Recurrent Neural network (RNN) Model :

```python
import tensorflow as tf
# Define the model architecture
model = tf.keras.Sequential([
    # Embedding layer to convert integer sequences into dense vectors
    tf.keras.layers.Embedding(input_dim=10000, output_dim=32, input_length=250),

    # Simple RNN layer
    tf.keras.layers.SimpleRNN(32, return_sequences=False),

    # Dense layer for binary classification
    tf.keras.layers.Dense(1, activation='sigmoid')
])# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
simple_rnn (SimpleRNN)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)

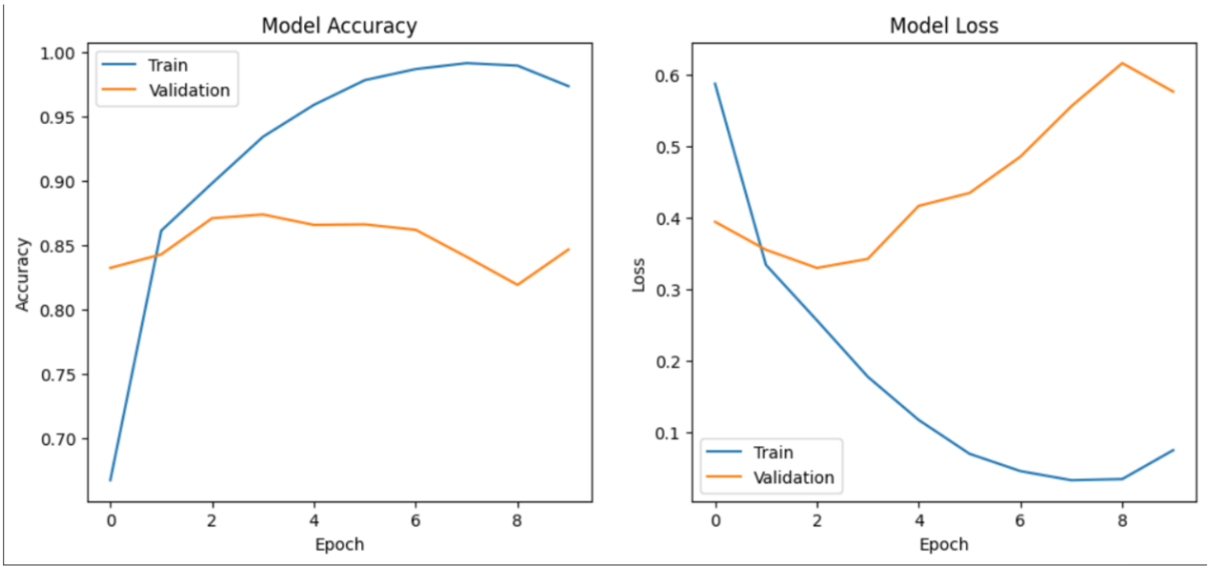
```
Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
```

(d) Training the RNN Model :

```python
# Define the number of epochs and batch size
epochs = 10
batch_size = 128
# Train the LSTM model
history_lstm = model_lstm.fit(
    X_train,  # Training features
    y_train,  # Training labels
    epochs=epochs,
    batch_size=batch_size,
    validation_data=(X_test, y_test),  # Validation data
    verbose=1  # Print progress during training
)
# Evaluate the model on the test dataset
test_loss_lstm, test_accuracy_lstm = model_lstm.evaluate(X_test, y_test, verbose=1)
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 5))
# Plot training & validation accuracy
plt.subplot(1, 2, 1)
plt.plot(history_lstm.history['accuracy'])
plt.plot(history_lstm.history['val_accuracy'])
plt.title('LSTM Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])
# Plot training & validation loss
plt.subplot(1, 2, 2)
plt.plot(history_lstm.history['loss'])
plt.plot(history_lstm.history['val_loss'])
plt.title('LSTM Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.show()
```

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))

# Plot training & validation accuracy
plt.subplot(1, 2, 1)
plt.plot(history_lstm.history['accuracy'])
plt.plot(history_lstm.history['val_accuracy'])
plt.title('LSTM Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])

# Plot training & validation loss
plt.subplot(1, 2, 2)
plt.plot(history_lstm.history['loss'])
plt.plot(history_lstm.history['val_loss'])
plt.title('LSTM Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])

plt.show()
```

```
Epoch 1/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 44s 119ms/step - accuracy: 0.5800 - loss: 0.6562 - val_accuracy: 0.8321 - val_loss: 0.3946
Epoch 2/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 30s 85ms/step - accuracy: 0.8494 - loss: 0.3561 - val_accuracy: 0.8426 - val_loss: 0.3555
Epoch 3/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 43s 91ms/step - accuracy: 0.9028 - loss: 0.2493 - val_accuracy: 0.8707 - val_loss: 0.3301
Epoch 4/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 42s 94ms/step - accuracy: 0.9377 - loss: 0.1724 - val_accuracy: 0.8736 - val_loss: 0.3427
Epoch 5/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 38s 84ms/step - accuracy: 0.9622 - loss: 0.1128 - val_accuracy: 0.8655 - val_loss: 0.4168
Epoch 6/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 85ms/step - accuracy: 0.9791 - loss: 0.0681 - val_accuracy: 0.8659 - val_loss: 0.4349
Epoch 7/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 84ms/step - accuracy: 0.9921 - loss: 0.0344 - val_accuracy: 0.8617 - val_loss: 0.4856
Epoch 8/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 84ms/step - accuracy: 0.9925 - loss: 0.0312 - val_accuracy: 0.8407 - val_loss: 0.5560
Epoch 9/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 83ms/step - accuracy: 0.9925 - loss: 0.0271 - val_accuracy: 0.8189 - val_loss: 0.6163
Epoch 10/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 41s 83ms/step - accuracy: 0.9814 - loss: 0.0544 - val_accuracy: 0.8464 - val_loss: 0.5765
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - accuracy: 0.8441 - loss: 0.5752
Test Loss: 0.5765175819396973
Test Accuracy: 0.8464000225067139
```



(e) Evaluate the model on the test dataset to see how well it generalizes.

```
# Evaluate the model on the test dataset
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=1)

# Print test results
print(f'Test Loss: {test_loss}')
print(f'Test Accuracy: {test_accuracy}')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - accuracy: 0.8441 - loss: 0.5752
Test Loss: 0.5765175819396973
Test Accuracy: 0.8464000225067139
```

2.Develop LSTM (Long Short-Term Memory) by utilizing data set from https://www.kaggle.com/code/amirrezaeian/time-series-data-analysis-using-lstm-tutorialLinks to an external site. or take any time series data. [CO2]

1. Build the LSTM Model

```python
import tensorflow as tf

# Define the LSTM model architecture
model_lstm = tf.keras.Sequential([
    # Embedding layer to convert integer sequences into dense vectors
    tf.keras.layers.Embedding(input_dim=10000, output_dim=32, input_length=250),

    # LSTM layer with 50 units
    tf.keras.layers.LSTM(50),

    # Dense layer for binary classification
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model_lstm.compile(optimizer='adam',
                   loss='binary_crossentropy',
                   metrics=['accuracy'])
model_lstm.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	?	0 (unbuilt)
lstm (LSTM)	?	0 (unbuilt)
dense_2 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

2. Train the LSTM Model:

```python
# Define the number of epochs and batch size
epochs = 10
batch_size = 128
# Train the LSTM model
history_lstm = model_lstm.fit(
    X_train,  # Training features
    y_train,  # Training labels
    epochs=epochs,
    batch_size=batch_size,
    validation_data=(X_test, y_test),  # Validation data
    verbose=1  # Print progress during training
)
```

```
Epoch 1/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 78s 242ms/step - accuracy: 0.6976 - loss: 0.5398 - val_accuracy: 0.8745 - val_loss: 0.2938
Epoch 2/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 73s 233ms/step - accuracy: 0.9055 - loss: 0.2455 - val_accuracy: 0.8857 - val_loss: 0.2788
Epoch 3/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 78s 221ms/step - accuracy: 0.9276 - loss: 0.1927 - val_accuracy: 0.8867 - val_loss: 0.2699
Epoch 4/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 68s 218ms/step - accuracy: 0.9438 - loss: 0.1560 - val_accuracy: 0.8905 - val_loss: 0.2813
Epoch 5/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 70s 223ms/step - accuracy: 0.9490 - loss: 0.1394 - val_accuracy: 0.8844 - val_loss: 0.3201
Epoch 6/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 80s 217ms/step - accuracy: 0.9600 - loss: 0.1141 - val_accuracy: 0.8781 - val_loss: 0.3526
Epoch 7/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 82s 217ms/step - accuracy: 0.9688 - loss: 0.0953 - val_accuracy: 0.8816 - val_loss: 0.3600
Epoch 8/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 72s 229ms/step - accuracy: 0.9614 - loss: 0.1050 - val_accuracy: 0.8761 - val_loss: 0.4356
Epoch 9/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 79s 220ms/step - accuracy: 0.9785 - loss: 0.0677 - val_accuracy: 0.8784 - val_loss: 0.4168
Epoch 10/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 66s 212ms/step - accuracy: 0.9800 - loss: 0.0635 - val_accuracy: 0.8736 - val_loss: 0.4035
```

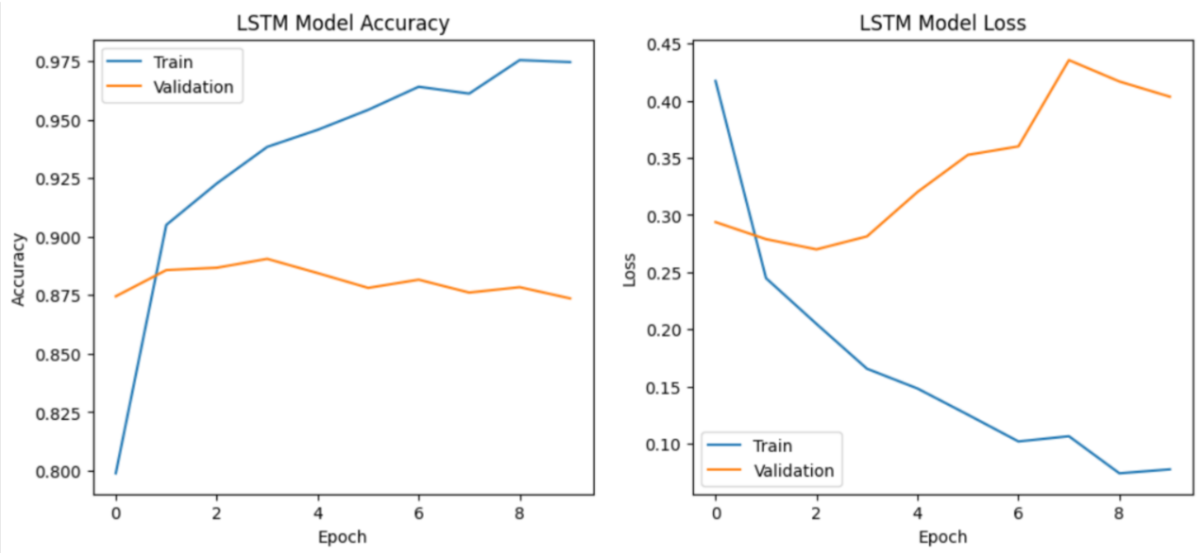
3. Evaluate the LSTM Model:

```
# Evaluate the model on the test dataset
test_loss_lstm, test_accuracy_lstm = model_lstm.evaluate(X_test, y_test, verbose=1)

# Print test results
print(f'Test Loss (LSTM): {test_loss_lstm}')
print(f'Test Accuracy (LSTM): {test_accuracy_lstm}')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 10s 31ms/step - accuracy: 0.8714 - loss: 0.4044
Test Loss (LSTM): 0.403506338596344
Test Accuracy (LSTM): 0.8736000061035156
```

4. Plot Training History:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 5))

# Plot training & validation accuracy
plt.subplot(1, 2, 1)
plt.plot(history_lstm.history['accuracy'])
plt.plot(history_lstm.history['val_accuracy'])
plt.title('LSTM Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'])

# Plot training & validation loss
plt.subplot(1, 2, 2)
plt.plot(history_lstm.history['loss'])
plt.plot(history_lstm.history['val_loss'])
plt.title('LSTM Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])

plt.show()
```

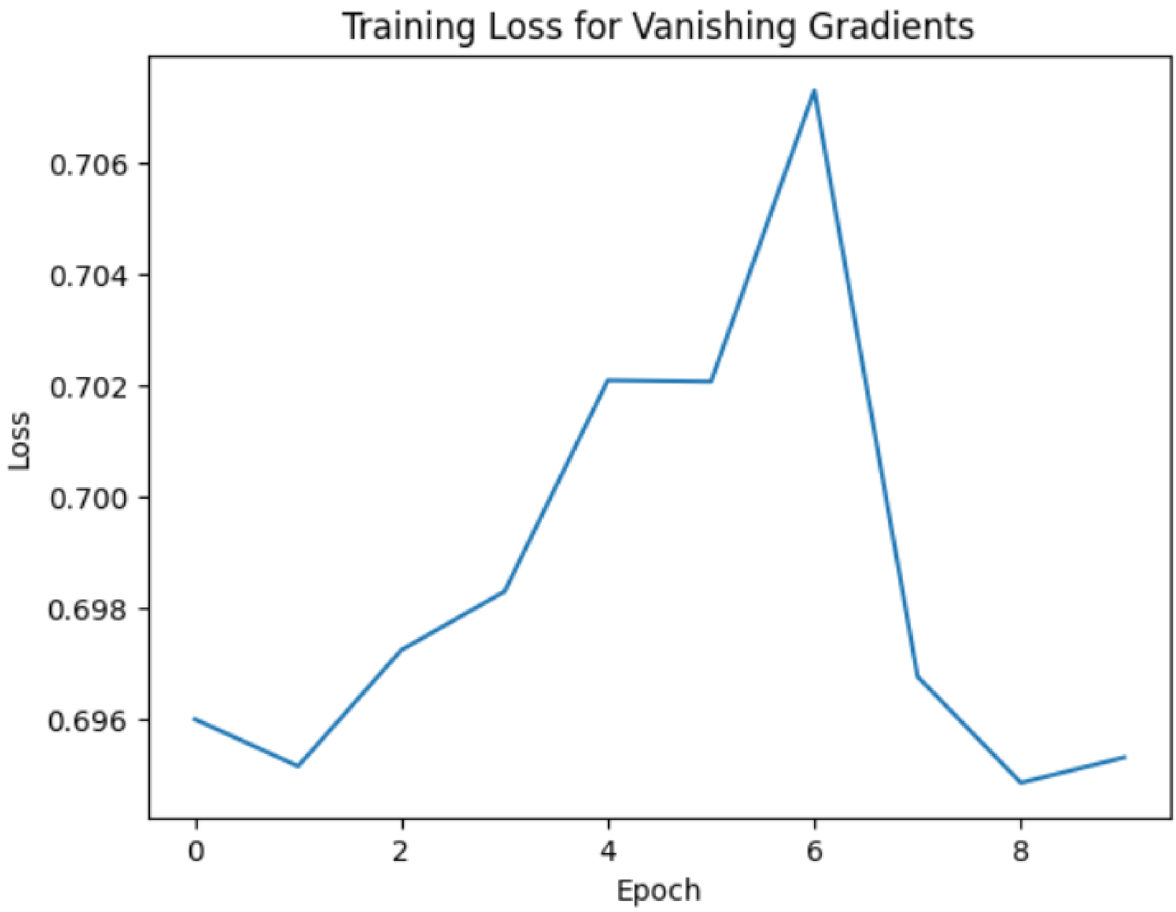


3.Demonstrate Vanishing and Exploding Gradients on deep neural network. [CO2]

1. Vanishing Gradients:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic data
X = np.random.rand(1000, 10)
y = np.random.randint(2, size=(1000, 1))

# Build a deep neural network with sigmoid activation
model_vanishing = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='sigmoid', input_shape=(10,)),
    tf.keras.layers.Dense(128, activation='sigmoid'),
    tf.keras.layers.Dense(128, activation='sigmoid'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model_vanishing.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history_vanishing = model_vanishing.fit(X, y, epochs=10, batch_size=32, verbose=1)

# Plot training loss
plt.plot(history_vanishing.history['loss'])
plt.title('Training Loss for Vanishing Gradients')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```

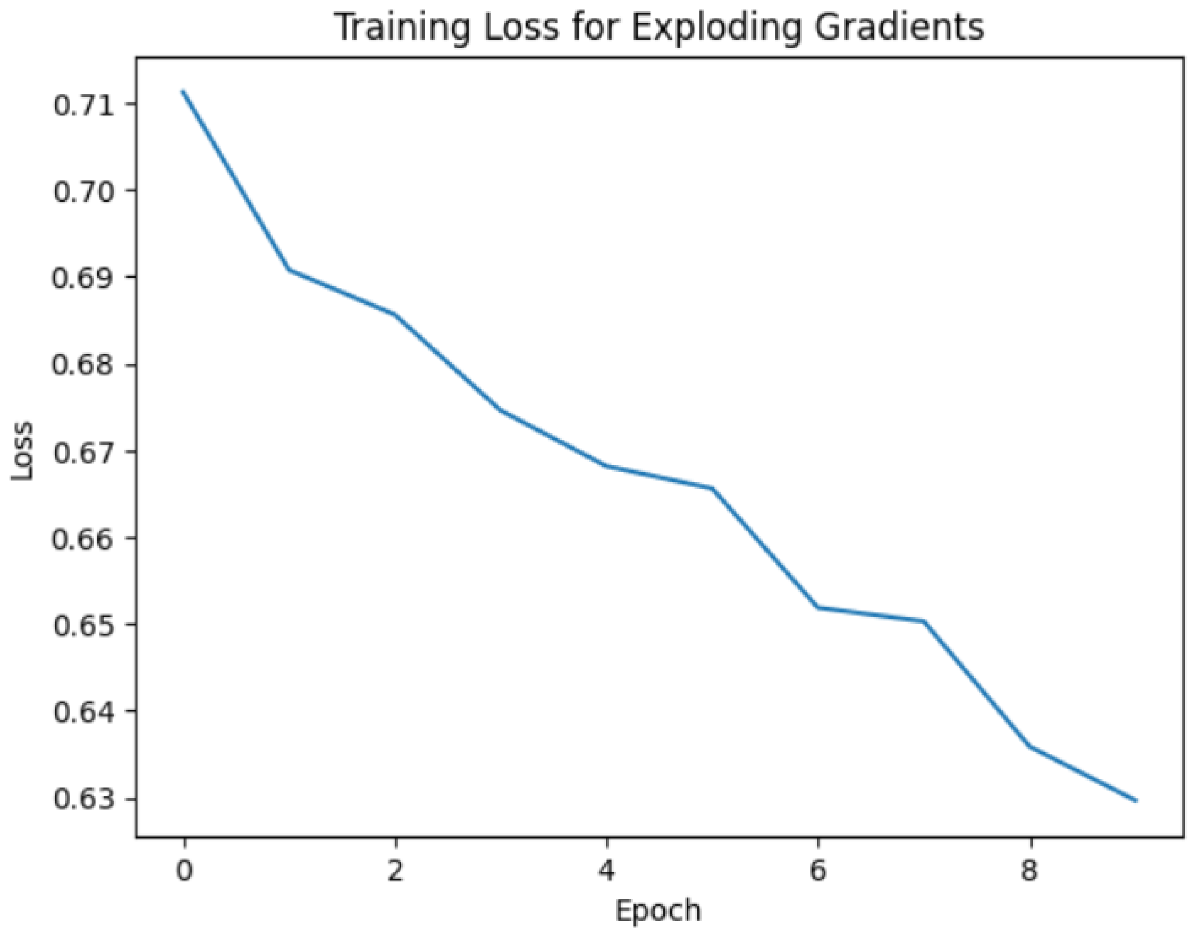
```
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.5028 - loss: 0.6958
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4814 - loss: 0.6952
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5158 - loss: 0.6954
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4967 - loss: 0.6985
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4992 - loss: 0.7025
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5107 - loss: 0.7010
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5080 - loss: 0.7015
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5356 - loss: 0.6961
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.4927 - loss: 0.6959
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4844 - loss: 0.7007
```



2. Exploding Gradients:

```python
# Generate synthetic data
X = np.random.rand(1000, 10)
y = np.random.randint(2, size=(1000, 1))

# Build a deep neural network with ReLU activation and large weight initialization
model_exploding = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(10,), kernel_initializer='he_normal'),
    tf.keras.layers.Dense(128, activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.Dense(128, activation='relu', kernel_initializer='he_normal'),
    tf.keras.layers.Dense(1, activation='sigmoid', kernel_initializer='he_normal')
])

# Compile the model
model_exploding.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history_exploding = model_exploding.fit(X, y, epochs=10, batch_size=32, verbose=1)

# Plot training loss
plt.plot(history_exploding.history['loss'])
plt.title('Training Loss for Exploding Gradients')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```

```
Epoch 1/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.5101 - loss: 0.7116
Epoch 2/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5384 - loss: 0.6879
Epoch 3/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5578 - loss: 0.6787
Epoch 4/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5373 - loss: 0.6777
Epoch 5/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5843 - loss: 0.6709
Epoch 6/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5892 - loss: 0.6656
Epoch 7/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6318 - loss: 0.6467
Epoch 8/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6237 - loss: 0.6501
Epoch 9/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6414 - loss: 0.6414
Epoch 10/10
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6685 - loss: 0.6220
```


Training Loss for Exploding Gradients
0.71
0.70
0.69
0.68
0.67
0.66
0.65
0.64
0.63
Loss
0
2
4
6
8
Epoch