finans sektöründe atılan tweetlerden duygu analizi yapılmaya çalışıldı.

```
#LSTM Tabanlı RNN Modeli ağırlık değerleri değiştirildi

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense,
Dropout, BatchNormalization, Bidirectional, LSTM
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau
from sklearn.metrics import accuracy_score, classification_report
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split

# Hiperparametreler
max_features = 10000  # Kelime sayısını artırarak modelin kelime
dağarcığını genişlet
max_len = 150  # Dizi uzunluğunu artırarak daha fazla bilgi kaydedin
embedding_dim = 128  # Gömme boyutunu optimize et
dropout_rate = 0.4  # Dropout oranını %40 olarak ayarla
learning_rate = 0.0005  # Öğrenme oranını küçük tutarak daha stabil
eğitim sağla


# Tokenizer tanımla ve veriyi dönüştür
tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(X_train)

# Eğitim ve test verilerini tam sayılara çevir ve sıfırlarla doldur
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)
X_train_pad = pad_sequences(X_train_seq, maxlen=max_len)
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len)

# One-hot encoding ile etiketleri dönüştür
y_train_onehot = to_categorical(y_train + 1, num_classes=3)  # -1 ->
0, 0 -> 1, 1 -> 2
y_test_onehot = to_categorical(y_test + 1, num_classes=3)

# Sınıf ağırlıklarını belirleyin (dengesiz sınıflar için)
class_weights = {0: 3., 1: 1., 2: 2.}  # -1 -> 0, 0 -> 1, 1 -> 2

# Modeli oluştur
model = Sequential([
    Embedding(max_features, embedding_dim, input_length=max_len,
```

```
trainable=True),
    Bidirectional(LSTM(128, activation='tanh', return_sequences=True,
kernel_initializer='he_normal', dropout=dropout_rate)),
    Bidirectional(LSTM(64, activation='tanh',
kernel_initializer='he_normal', dropout=dropout_rate)),
    BatchNormalization(),  # BatchNormalization ekleyin
    Dropout(dropout_rate),
    Dense(3, activation='softmax')  # Softmax aktivasyon ile çoklu
sınıf çıkışı
])

# Öğrenme oranını ayarlayın
optimizer = Adam(learning_rate=learning_rate)

# Modeli derle
model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])

# Erken durdurma ile eğitimi durdurma ve Öğrenme oranı düşürme
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
lr_scheduler = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=3, min_lr=1e-6)

# Modeli eğit
history = model.fit(
    X_train_pad, y_train_onehot,  # y_train_onehot çoklu sınıf
etiketlerinin one-hot kodlanmış hali
    epochs=60, batch_size=32,
    validation_split=0.2,
    class_weight=class_weights,  # Sınıf ağırlıklarını ekle
    callbacks=[early_stopping, lr_scheduler]
)

# Test verileri üzerinde tahmin yap
predictions_rnn = np.argmax(model.predict(X_test_pad), axis=1)  #
Çoklu sınıf tahmini

# Doğruluk skorunu hesapla
accuracy = accuracy_score(np.argmax(y_test_onehot, axis=1),
predictions_rnn)
print(f"RNN Accuracy Score -> {accuracy * 100:.2f}%")

# Detaylı metrik raporu (etiketleri -1, 0 ve 1'e dönüştür)
predictions_rnn = predictions_rnn - 1  # 0 -> -1, 1 -> 0, 2 -> 1
y_test_actual = np.argmax(y_test_onehot, axis=1) - 1  # 0 -> -1, 1 ->
0, 2 -> 1
print(classification_report(y_test_actual, predictions_rnn))

# Eğitim süreci doğrulama doğruluğu ve doğrulama kaybı eğrisini çizme
```

```python
plt.figure(figsize=(12, 6))

# Doğrulama doğruluğu
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Doğrulama kaybı
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Epoch 1/60
438/438 ━━━━━━━━━━━━━━━━━━━━ 505s 1s/step - accuracy: 0.4370 - loss:
2.1613 - val_accuracy: 0.7224 - val_loss: 0.7577 - learning_rate:
5.0000e-04
Epoch 2/60
438/438 ━━━━━━━━━━━━━━━━━━━━ 500s 1s/step - accuracy: 0.7769 - loss:
0.9377 - val_accuracy: 0.7699 - val_loss: 0.5824 - learning_rate:
5.0000e-04
Epoch 3/60
438/438 ━━━━━━━━━━━━━━━━━━━━ 503s 1s/step - accuracy: 0.8454 - loss:
0.6814 - val_accuracy: 0.8090 - val_loss: 0.5621 - learning_rate:
5.0000e-04
Epoch 4/60
438/438 ━━━━━━━━━━━━━━━━━━━━ 500s 1s/step - accuracy: 0.8832 - loss:
0.5126 - val_accuracy: 0.8036 - val_loss: 0.5969 - learning_rate:
5.0000e-04
Epoch 5/60
438/438 ━━━━━━━━━━━━━━━━━━━━ 538s 1s/step - accuracy: 0.9017 - loss:
0.4409 - val_accuracy: 0.8005 - val_loss: 0.6398 - learning_rate:
5.0000e-04
Epoch 6/60
438/438 ━━━━━━━━━━━━━━━━━━━━ 523s 1s/step - accuracy: 0.9149 - loss:
0.3945 - val_accuracy: 0.7965 - val_loss: 0.7067 - learning_rate:
5.0000e-04
Epoch 7/60
438/438 ━━━━━━━━━━━━━━━━━━━━ 490s 1s/step - accuracy: 0.9380 - loss:
0.3009 - val_accuracy: 0.8153 - val_loss: 0.7185 - learning_rate:
```

```
2.5000e-04
Epoch 8/60
438/438 ━━━━━━━━━━━━━━━━━━━━ 490s 1s/step - accuracy: 0.9547 - loss:
0.2448 - val_accuracy: 0.8062 - val_loss: 0.7541 - learning_rate:
2.5000e-04
137/137 ━━━━━━━━━━━━━━━━━━━━ 45s 323ms/step
RNN Accuracy Score -> 79.14%
              precision    recall  f1-score   support

          -1       0.70      0.79      0.74       948
           0       0.85      0.81      0.83      1988
           1       0.79      0.76      0.77      1437

    accuracy                           0.79      4373
   macro avg       0.78      0.79      0.78      4373
weighted avg       0.80      0.79      0.79      4373
```