import tensorflow as tf

from tensorflow.keras.datasets import imdb

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

# Load the IMDB dataset

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=10000)

# Preprocess the data

maxlen = 500  # Maximum number of words in a review

x\_train = tf.keras.preprocessing.sequence.pad\_sequences(x\_train, maxlen=maxlen)

x\_test = tf.keras.preprocessing.sequence.pad\_sequences(x\_test, maxlen=maxlen)

# Build the model

model = Sequential()

model.add(Embedding(10000, 32))

model.add(SimpleRNN(32))

model.add(Dense(1, activation='sigmoid'))

# Compile the model

model.compile(optimizer='rmsprop', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(x\_train, y\_train, epochs=10, batch\_size=128, validation\_split=0.2)

# Evaluate the model

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print('Test accuracy:', test\_acc)

**Kaggle code for LSTM embedding:**

https://www.kaggle.com/code/mdjohirulislam/using-lstm-sentiment-analysis#Word-Embedding

import torch

import torch.nn as nn

import torch.optim as optim

from torchtext.datasets import IMDB

from torchtext.data import Field, LabelField, BucketIterator

from sklearn.metrics import accuracy\_score

# define the device to use

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

# define the fields for preprocessing the data

text\_field = Field(tokenize='spacy', lower=True)

label\_field = LabelField(dtype=torch.float)

# download and split the IMDB dataset

train\_data, test\_data = IMDB.splits(text\_field, label\_field)

# build the vocabulary

text\_field.build\_vocab(train\_data, max\_size=25000, vectors='glove.6B.100d')

label\_field.build\_vocab(train\_data)

# create the data iterators

BATCH\_SIZE = 64

train\_loader, test\_loader = BucketIterator.splits((train\_data, test\_data), batch\_size=BATCH\_SIZE, device=device)

# define the LSTM model

class LSTMModel(nn.Module):

    def \_\_init\_\_(self, input\_dim, embedding\_dim, hidden\_dim, output\_dim, num\_layers, dropout):

        super().\_\_init\_\_()

        self.embedding = nn.Embedding(input\_dim, embedding\_dim)

        self.lstm = nn.LSTM(embedding\_dim, hidden\_dim, num\_layers, dropout=dropout, batch\_first=True)

        self.fc = nn.Linear(hidden\_dim, output\_dim)

        self.dropout = nn.Dropout(dropout)

    def forward(self, text):

        embedded = self.embedding(text)

        embedded = self.dropout(embedded)

        output, (hidden, cell) = self.lstm(embedded)

        hidden = self.dropout(hidden[-1])

        return self.fc(hidden)

# define the model hyperparameters

EMBEDDING\_DIM = 100

HIDDEN\_DIM = 256

OUTPUT\_DIM = 1

NUM\_LAYERS = 2

DROPOUT = 0.5

# create an instance of the model

model = LSTMModel(len(text\_field.vocab), EMBEDDING\_DIM, HIDDEN\_DIM, OUTPUT\_DIM, NUM\_LAYERS, DROPOUT).to(device)

# initialize the embeddings with pre-trained GloVe vectors

pretrained\_embeddings = text\_field.vocab.vectors

model.embedding.weight.data.copy\_(pretrained\_embeddings)

# define the optimizer and loss function

optimizer = optim.Adam(model.parameters())

criterion = nn.BCEWithLogitsLoss()

# define the training loop

def train(model, train\_loader, optimizer, criterion):

    epoch\_loss = 0

    epoch\_acc = 0

    model.train()

    for batch in train\_loader:

        optimizer.zero\_grad()

        inputs = batch.text.to(device)

        labels = batch.label.to(device)

        preds = model(inputs).squeeze()

        loss = criterion(preds, labels)

        acc = accuracy\_score(labels.detach().cpu().numpy(), torch.sigmoid(preds).detach().cpu().numpy().round())

        loss.backward()

        optimizer.step()

        epoch\_loss += loss.item()

        epoch\_acc += acc.item()

    return epoch\_loss / len(train\_loader), epoch\_acc / len(train\_loader)

# define the testing loop

def test(model, test\_loader, criterion):

    epoch\_loss = 0

    epoch\_acc = 0

    model.eval()

    with torch.no\_grad():

        for batch in test\_loader:

            inputs = batch.text.to(device)

            labels = batch.label.to(device)

            preds = model(inputs).squeeze()

            loss = criterion(preds, labels)

            acc = accuracy\_score(labels.detach().cpu().numpy(), torch.sigmoidpreds.detach().cpu().numpy().round())

            epoch\_loss += loss.item()

            epoch\_acc += acc.item()

    return epoch\_loss / len(test\_loader), epoch\_acc / len(test\_loader)

# train and test the model

NUM\_EPOCHS = 10

for epoch in range(NUM\_EPOCHS):

    train\_loss, train\_acc = train(model, train\_loader, optimizer, criterion)

    test\_loss, test\_acc = test(model, test\_loader, criterion)

    print(f'Epoch: {epoch+1:02}\n\tTrain Loss: {train\_loss:.3f} | Train Acc: {train\_acc\*100:.2f}%')

    print(f'\tTest Loss: {test\_loss:.3f} | Test Acc: {test\_acc\*100:.2f}%')

def evaluate(model, test\_loader):

    true\_labels = []

    pred\_labels = []

    model.eval()

    with torch.no\_grad():

        for batch in test\_loader:

            inputs = batch.text.to(device)

            labels = batch.label.to(device)

            preds = model(inputs).squeeze()

            true\_labels += labels.detach().cpu().numpy().tolist()

            pred\_labels += torch.sigmoid(preds).detach().cpu().numpy().round().tolist()

    acc = accuracy\_score(true\_labels, pred\_labels)

    return acc

test\_acc = evaluate(model, test\_loader)

print(f'Test accuracy: {test\_acc\*100:.2f}%')