```
#installation of some libraries
!pip install --upgrade torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cpu -q
!pip install datasets -q
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 178.7/178.7 MB 4.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 17.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 16.1 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sou
fastai 2.7.18 requires torch<2.6,>=1.10, but you have torch 2.6.0+cpu which is incompatible.
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 491.2/491.2 kB 9.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 6.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 143.5/143.5 kB 7.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.8/194.8 kB 13.7 MB/s eta 0:00:00
```

```
##importing libraries
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import matplotlib.pyplot as plt
import random
import numpy as np


#setting a random seed
def set_seed(seed=42):
    torch.manual_seed(seed)
    np.random.seed(seed)
    random.seed(seed)

set_seed()


#checking the distribution of dataset
from collections import Counter
print("Train:", Counter([int(d['labels']) for d in train_dataset]))
print("Test:", Counter([int(d['labels']) for d in test_dataset]))
```

```
Train: Counter({1: 514, 0: 486})
Test: Counter({0: 101, 1: 99})
```

```
#Converting train_dataset and test_dataset as PyTorch Dataset objects
class DummySentimentDataset(Dataset):
    def __init__(self, vocab_size, num_samples=1000):
        self.vocab_size = vocab_size
        self.num_samples = num_samples
        self.data = [
            {
                'input_ids': torch.randint(0, vocab_size, (10,)),
                'labels': torch.tensor(float(random.randint(0, 1)))
            }
            for _ in range(num_samples)
        ]

    def __len__(self):
        return self.num_samples

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

# Define vocab and datasets
vocab_size = 1000
train_dataset = DummySentimentDataset(vocab_size, 1000)
test_dataset = DummySentimentDataset(vocab_size, 200)



#Defining model for testing hyperparameters
class LinearSentimentClassifier(nn.Module):
    def __init__(self, vocab_size, embedding_dim=10, dropout_prob=0.3):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.dropout = nn.Dropout(dropout_prob)
        self.linear = nn.Linear(embedding_dim, 1)
        self.bias = nn.Parameter(torch.zeros(1))
        nn.init.uniform_(self.embedding.weight, -0.1, 0.1)

    def forward(self, input_ids):
        embedded = self.embedding(input_ids).sum(dim=1)  # [batch_size, embedding_dim]
        embedded = self.dropout(embedded)
        logits = self.linear(embedded) + self.bias  # [batch_size, 1]
        return torch.sigmoid(logits)  # still [batch_size, 1]



# Splitting the data into training and eveaulating

from torch.utils.data import random_split

def split_dataset(dataset, val_ratio=0.2):
    val_size = int(len(dataset) * val_ratio)
    train_size = len(dataset) - val_size
    return random_split(dataset, [train_size, val_size])



#defining the model for training
def train_model(vocab_size, full_train_dataset, test_dataset,
                learning_rate=1e-3, batch_size=32, embedding_dim=50,
                dropout_prob=0.5, num_epochs=10, val_ratio=0.2,
                early_stopping=True, patience=5):

    # Split train into train + val
    train_dataset, val_dataset = split_dataset(full_train_dataset, val_ratio)
    print(f"Training on {len(train_dataset)} samples, validating on {len(val_dataset)}")

    # Data loaders
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=64)
    test_loader = DataLoader(test_dataset, batch_size=64)

    # Model and optimizer
    model = LinearSentimentClassifier(vocab_size, embedding_dim, dropout_prob)
    criterion = nn.BCELoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    train_losses, train_accuracies = [], []
    val_accuracies = []

    best_val_acc = 0
    patience_counter = 0

    for epoch in range(num_epochs):
        # ---- Training ----
        model.train()
        total_loss, correct, total = 0.0, 0, 0
        for batch in train_loader:
            inputs = batch['input_ids']
            labels = batch['labels'].float()
            outputs = model(inputs).squeeze(1)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()

            preds = (outputs >= 0.5).float()
            correct += (preds == labels).sum().item()
            total += labels.size(0)
            total_loss += loss.item()

        train_acc = correct / total * 100
        train_losses.append(total_loss / len(train_loader))
        train_accuracies.append(train_acc)

        # ---- Validation ----
        model.eval()
        correct, total = 0, 0
        with torch.no_grad():
            for batch in val_loader:
                inputs = batch['input_ids']
                labels = batch['labels'].float()
                outputs = model(inputs).squeeze(1)
                preds = (outputs >= 0.5).float()
                correct += (preds == labels).sum().item()
                total += labels.size(0)

        val_acc = correct / total * 100
        val_accuracies.append(val_acc)

        print(f"Epoch {epoch+1}/{num_epochs}: "
              f"Train Acc = {train_acc:.2f}%, Val Acc = {val_acc:.2f}%")
        # --- # Early stopping #----#
        if early_stopping:
            if val_acc > best_val_acc:
                best_val_acc = val_acc
                patience_counter = 0
            else:
                patience_counter += 1
                if patience_counter >= patience:
                    print("⚠️ Early stopping triggered.")
                    break

    # ---- Final Test Evaluation ----
    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for batch in test_loader:
            inputs = batch['input_ids']
            labels = batch['labels'].float()
            outputs = model(inputs).squeeze(1)
            preds = (outputs >= 0.5).float()
            correct += (preds == labels).sum().item()
            total += labels.size(0)

    test_acc = correct / total * 100
    print(f"\n✅ Final Test Accuracy: {test_acc:.2f}%")

    return test_acc



#training the model
train_model(
    vocab_size=vocab_size,
    full_train_dataset=train_dataset,
    test_dataset=test_dataset,
    learning_rate=1e-3,
    batch_size=32,
    embedding_dim=50,
    dropout_prob=0.5,
    num_epochs=10
)
```

```
Training on 800 samples, validating on 200
Epoch 1/10: Train Acc = 50.62%, Val Acc = 51.00%
Epoch 2/10: Train Acc = 60.00%, Val Acc = 51.00%
Epoch 3/10: Train Acc = 70.00%, Val Acc = 53.00%
Epoch 4/10: Train Acc = 75.12%, Val Acc = 54.50%
Epoch 5/10: Train Acc = 77.12%, Val Acc = 54.00%
Epoch 6/10: Train Acc = 81.50%, Val Acc = 53.50%
Epoch 7/10: Train Acc = 83.88%, Val Acc = 54.50%
Epoch 8/10: Train Acc = 85.00%, Val Acc = 53.00%
Epoch 9/10: Train Acc = 86.88%, Val Acc = 53.50%
⚠️ Early stopping triggered.

✅ Final Test Accuracy: 48.50%
48.5
```

Start coding or generate with AI.

```
Train: Counter({1: 514, 0: 486})
Test: Counter({0: 101, 1: 99})
```

Start coding or generate with AI.