```
!pip uninstall -y torch torchvision torchaudio
!pip install torch==2.0.1 torchvision==0.15.2 torchaudio==2.0.2 --index-url https://download.pytorch.org/whl/cpu
!pip install datasets -q
```

```
Found existing installation: torch 2.6.0+cu124
Uninstalling torch-2.6.0+cu124:
  Successfully uninstalled torch-2.6.0+cu124
Found existing installation: torchvision 0.21.0+cu124
Uninstalling torchvision-0.21.0+cu124:
  Successfully uninstalled torchvision-0.21.0+cu124
Found existing installation: torchaudio 2.6.0+cu124
Uninstalling torchaudio-2.6.0+cu124:
  Successfully uninstalled torchaudio-2.6.0+cu124
Looking in indexes: https://download.pytorch.org/whl/cpu
Collecting torch==2.0.1
  Downloading https://download.pytorch.org/whl/cpu/torch-2.0.1%2Bcpu-cp311-cp311-linux_x86_64.whl (195.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 195.4/195.4 MB 5.6 MB/s eta 0:00:00
Collecting torchvision==0.15.2
  Downloading https://download.pytorch.org/whl/cpu/torchvision-0.15.2%2Bcpu-cp311-cp311-linux_x86_64.whl (1.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.5/1.5 MB 39.5 MB/s eta 0:00:00
Collecting torchaudio==2.0.2
  Downloading https://download.pytorch.org/whl/cpu/torchaudio-2.0.2%2Bcpu-cp311-cp311-linux_x86_64.whl (4.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.1/4.1 MB 48.5 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch==2.0.1) (3.18.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from torch==2.0.1) (4.13.1)
Requirement already satisfied: sympy in /usr/local/lib/python3.11/dist-packages (from torch==2.0.1) (1.13.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch==2.0.1) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch==2.0.1) (3.1.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from torchvision==0.15.2) (2.0.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from torchvision==0.15.2) (2.32.3)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.11/dist-packages (from torchvision==0.15.2) (11.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch==2.0.1) (3.0.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->torchvision==0.15
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->torchvision==0.15.2) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->torchvision==0.15.2) (2
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->torchvision==0.15.2) (2
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy->torch==2.0.1) (1.3.0)
Installing collected packages: torch, torchvision, torchaudio
Successfully installed torch-2.0.1+cpu torchaudio-2.0.2+cpu torchvision-0.15.2+cpu
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 491.2/491.2 kB 26.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 9.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 183.9/183.9 kB 13.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 143.5/143.5 kB 9.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 194.8/194.8 kB 14.5 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
gcsfs 2025.3.2 requires fsspec==2025.3.2, but you have fsspec 2024.12.0 which is incompatible.
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from datasets import load_dataset
from sklearn.metrics import accuracy_score
from collections import Counter
from itertools import chain
import numpy as np


# Load the IMDB dataset
imdb = load_dataset("imdb")


# Improved tokenizer with basic preprocessing (lowercasing, alphabetic filtering)
def simple_tokenizer(text):
    text = text.lower()
    tokens = re.findall(r'\b[a-z]{3,}\b', text)  # keep only words with 3+ letters
    return tokens


# Filter vocabulary with frequency cutoff
MIN_FREQ = 5
counter = Counter(chain.from_iterable(simple_tokenizer(example['text']) for example in imdb['train']))
filtered = {word: freq for word, freq in counter.items() if freq >= MIN_FREQ}
vocab = {word: i+2 for i, word in enumerate(filtered)}  # +2 for pad and unk
vocab['[PAD]'] = 0
vocab['[UNK]'] = 1
inv_vocab = {i: w for w, i in vocab.items()}


# Tokenization and Encoding
def encode(example):
    tokens = simple_tokenizer(example['text'])
    ids = [vocab.get(token, vocab['[UNK]']) for token in tokens]
    return {'input_ids': ids, 'label': example['label']}
```

```
tokenized_imdb = imdb.map(encode, remove_columns=['text'])
```

```
Map: 100%  25000/25000 [00:08<00:00, 2410.54 examples/s]
Map: 100%  25000/25000 [00:04<00:00, 5044.94 examples/s]
Map: 100%  50000/50000 [00:11<00:00, 4621.16 examples/s]
```

```
# Padding
def pad(batch):
    max_len = max(len(x) for x in batch['input_ids'])
    padded = [x + [vocab['[PAD]']] * (max_len - len(x)) for x in batch['input_ids']]
    return {'input_ids': torch.tensor(padded), 'labels': torch.tensor(batch['label'])}


# Dataset Wrapper
from torch.utils.data import Dataset, DataLoader

class IMDbDataset(Dataset):
    def __init__(self, encodings):
        self.encodings = encodings

    def __getitem__(self, idx):
        item = {
            'input_ids': self.encodings['input_ids'][idx],
            'labels': self.encodings['labels'][idx]
        }
        return item

    def __len__(self):
        return len(self.encodings['input_ids'])


# Pad and wrap datasets
train_dataset = pad(tokenized_imdb['train'])
test_dataset = pad(tokenized_imdb['test'])
train_dataset = IMDbDataset(train_dataset)
test_dataset = IMDbDataset(test_dataset)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64)


# Linear Classifier
class LinearSentimentClassifier(nn.Module):
    def __init__(self, vocab_size):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, 1)

    def forward(self, input_ids):
        embedded = self.embedding(input_ids).sum(dim=1).squeeze(1)
        return torch.sigmoid(embedded)


# Updated Model with Bias and Init Fixes
class LinearSentimentClassifier(nn.Module):
    def __init__(self, vocab_size):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, 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).squeeze(1)
        logits = embedded + self.bias
        return torch.sigmoid(logits)

# Train Function with Debug Logging
model = LinearSentimentClassifier(len(vocab))
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters())

def train():
    model.train()
    for epoch in range(4):
        total_loss = 0.0
        correct = 0
        total = 0
        for i, batch in enumerate(train_loader):
            optimizer.zero_grad()
            inputs = batch['input_ids']
            labels = batch['labels'].float()
```

```
            # Clamp to avoid index errors
            inputs = torch.clamp(inputs, max=len(vocab)-1)

            outputs = model(inputs)

            # Binary predictions for accuracy
            preds = (outputs >= 0.5).float()
            correct += (preds == labels).sum().item()
            total += labels.size(0)

            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()

            # Debug info every 200 batches
            if i % 200 == 0:
                print(f"  Batch {i}, Loss: {loss.item():.4f}, Outputs avg: {outputs.mean().item():.4f}, Labels avg: {labels.mean().item()

        avg_loss = total_loss / len(train_loader)
        accuracy = correct / total * 100
        print(f"Epoch {epoch+1}, Avg Loss: {avg_loss:.4f}, Accuracy: {accuracy:.2f}%")

train()
```

```
  Batch 0, Loss: 31.2500, Outputs avg: 0.0000, Labels avg: 0.3125
  Batch 200, Loss: 50.0000, Outputs avg: 0.0000, Labels avg: 0.5000
  Batch 400, Loss: 15.2834, Outputs avg: 0.0000, Labels avg: 0.4688
  Batch 600, Loss: 0.6059, Outputs avg: 0.4438, Labels avg: 0.6562
Epoch 1, Avg Loss: 25.0052, Accuracy: 60.81%
  Batch 0, Loss: 0.3579, Outputs avg: 0.3212, Labels avg: 0.4688
  Batch 200, Loss: 0.2883, Outputs avg: 0.3558, Labels avg: 0.4062
  Batch 400, Loss: 0.4662, Outputs avg: 0.3537, Labels avg: 0.5938
  Batch 600, Loss: 0.2996, Outputs avg: 0.3991, Labels avg: 0.4688
Epoch 2, Avg Loss: 0.3277, Accuracy: 86.90%
  Batch 0, Loss: 0.1835, Outputs avg: 0.4832, Labels avg: 0.4688
  Batch 200, Loss: 0.2089, Outputs avg: 0.4151, Labels avg: 0.3750
  Batch 400, Loss: 0.1718, Outputs avg: 0.4170, Labels avg: 0.4062
  Batch 600, Loss: 0.1351, Outputs avg: 0.5907, Labels avg: 0.5938
Epoch 3, Avg Loss: 0.2329, Accuracy: 91.91%
  Batch 0, Loss: 0.1644, Outputs avg: 0.4386, Labels avg: 0.4375
  Batch 200, Loss: 0.2499, Outputs avg: 0.6591, Labels avg: 0.5312
  Batch 400, Loss: 0.1383, Outputs avg: 0.3974, Labels avg: 0.4688
  Batch 600, Loss: 0.1787, Outputs avg: 0.5330, Labels avg: 0.5000
Epoch 4, Avg Loss: 0.1851, Accuracy: 94.10%
```

```
# Word importance analysis
weights = model.embedding.weight.data.squeeze()
word_weights = [(inv_vocab[i], weights[i].item()) for i in range(len(weights)) if i in inv_vocab]
sorted_words = sorted(word_weights, key=lambda x: x[1])

print("\n🔻 Most negative words:")
print([w for w, _ in sorted_words[:20]])

print("\n🔺 Most positive words:")
print([w for w, _ in sorted_words[-20:]])
```

```
🔻 Most negative words:
['manage', 'obvious', 'gesture', 'beautiful', 'role', 'dozens', 'all!', 'competence', 'call', 'War.<br', 'told.', 'talented', "ain't

🔺 Most positive words:
['ordinary', 'task.<br', 'disappointed', 'it:', "Foxx's", 'majority', 'utterly', 'chronicle', 'boring', 'minutes.', 'dang', 'most',
```

Start coding or generate with AI.