```
# Install required packages via pip
!pip install spacy nltk torch torchcrf scikit-learn
# Download the spaCy English model
!pip python -m spacy download en_core_web_sm
!python -c "import nltk; nltk.download('treebank')"
!pip install pytorch-crf
import spacy
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torchcrf import CRF
from nltk.corpus import treebank
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader, Dataset

# Load spaCy model
nlp = spacy.load("en_core_web_sm")

# Load Treebank dataset
sentences = treebank.sents()
tags = [[tag for _, tag in sent] for sent in treebank.tagged_sents()]


# Build vocabulary
word2idx = {word: idx + 1 for idx, word in enumerate(set(word for sent in sentences
for word in sent))}
tag2idx = {tag: idx for idx, tag in enumerate(set(tag for sent in tags for tag in
sent))}
word2idx["<PAD>"] = 0
idx2tag = {idx: tag for tag, idx in tag2idx.items()}


# Convert data to indices
X = [[word2idx[word] for word in sent] for sent in sentences]
y = [[tag2idx[tag] for tag in sent] for sent in tags]
4
# Padding sequences
max_len = max(len(sent) for sent in X)
X = [sent + [0] * (max_len - len(sent)) for sent in X]
y = [sent + [tag2idx['NN']] * (max_len - len(sent)) for sent in y]  # NN as default
padding label

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Convert to PyTorch tensors
X_train, X_test = torch.tensor(X_train, dtype=torch.long), torch.tensor(X_test,
dtype=torch.long)
```

```
y_train, y_test = torch.tensor(y_train, dtype=torch.long), torch.tensor(y_test,
dtype=torch.long)


# Dataset class
class POSTaggingDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

    def __len__(self):
        return len(self.X)

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

train_dataset = POSTaggingDataset(X_train, y_train)
test_dataset = POSTaggingDataset(X_test, y_test)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
# Bi-LSTM with CRF model
class BiLSTM_CRF(nn.Module):
    def __init__(self, vocab_size, tagset_size, embedding_dim=128, hidden_dim=256):
        super(BiLSTM_CRF, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim // 2, num_layers=2,
bidirectional=True, batch_first=True)
        self.fc = nn.Linear(hidden_dim, tagset_size)
        self.crf = CRF(tagset_size, batch_first=True)

    def forward(self, x, tags=None):
        x = self.embedding(x)
        x, _ = self.lstm(x)
        emissions = self.fc(x)
        # Create a mask with the correct shape
        mask = (x[:, :, 0] != 0)  # Assume padding token is 0 and is the first
dimension in embedding

        if tags is not None:
            return -self.crf(emissions, tags, mask=mask, reduction='mean')
        else:
            return self.crf.decode(emissions, mask=mask) # Also apply mask during
decoding

# Model, optimizer, loss
model = BiLSTM_CRF(len(word2idx), len(tag2idx))
optimizer = optim.Adam(model.parameters(), lr=0.001)
# Training
num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    for X_batch, y_batch in train_loader:
        optimizer.zero_grad()
        loss = model(X_batch, y_batch)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print(f"Epoch {epoch+1}/{num_epochs}, Loss: {total_loss / len(train_loader)}")
# Separate cell for evaluating accuracy on the test set
model.eval()  # Set model to evaluation mode
total_correct = 0
total_tokens = 0

with torch.no_grad():
    for X_batch, y_batch in test_loader:
        predictions = model(X_batch)  # Returns a list of lists with predicted tag indices
        for pred_seq, true_seq in zip(predictions, y_batch.tolist()):
            # Compare each token's predicted tag with the true tag
            for pred, true in zip(pred_seq, true_seq):
                total_correct += (pred == true)
                total_tokens += 1

accuracy = (total_correct / total_tokens)*100
print(f"Accuracy on test set: {accuracy:.4f}")
from sklearn.metrics import classification_report

# Collect true and predicted labels
true_labels = []
predicted_labels = []

model.eval()
with torch.no_grad():
    for X_batch, y_batch in test_loader:
        predictions = model(X_batch)  # Get predicted indices
        for pred_seq, true_seq in zip(predictions, y_batch.tolist()):
            true_labels.extend(true_seq)
            predicted_labels.extend(pred_seq)

# Convert indices back to POS tags
true_labels = [idx2tag[idx] for idx in true_labels]
predicted_labels = [idx2tag[idx] for idx in predicted_labels]

# Compute and print F1-score, Precision, Recall
print(classification_report(true_labels, predicted_labels))
import gradio as gr
import torch
import spacy
```

```
# Load spaCy model
nlp = spacy.load("en_core_web_sm")

# Assume word2idx, idx2tag, and model are already defined and loaded

def predict_pos_tags(user_input):
    # Tokenize the input
    doc = nlp(user_input)
    tokens = [token.text for token in doc]

    # Convert tokens to indices
    token_ids = [word2idx.get(token, 0) for token in tokens]

    # Create tensor with batch dimension
    input_tensor = torch.tensor([token_ids], dtype=torch.long)

    # Set model to evaluation mode and get predictions
    model.eval()
    with torch.no_grad():
        predictions = model(input_tensor)
    predicted_tag_indices = predictions[0]

    # Convert predicted indices back to POS tag labels
    predicted_tags = [idx2tag[idx] for idx in predicted_tag_indices]

    # Format the output
    result = "Token\tPredicted POS Tag\n"
    result += "\n".join([f"{token}\t{tag}" for token, tag in zip(tokens,
predicted_tags)])
    return result

# Create Gradio UI
interface = gr.Interface(
    fn=predict_pos_tags,
    inputs=gr.Textbox(lines=2, placeholder="Enter your sentence here..."),
    outputs="text",
    title="POS Tagger",
    description="Enter a sentence to get POS tags predicted by the BiLSTM-CRF
model"
)

# Launch Gradio app
interface.launch()
```