```
from transformers import AutoTokenizer
import os
from torch.utils.data import DataLoader, Dataset
import torch
import numpy as np
import torch.nn as nn
from transformers import AutoModel
import torch.optim as optim
import torch.nn.functional as F

VOCAB = ('<PAD>', 'O', 'B-Chemical', 'B-Disease', 'I-Disease', 'I-
Chemical')
tag2idx = {v: k for k, v in enumerate(VOCAB)}

tokenizer =
AutoTokenizer.from_pretrained("monologg/biobert_v1.0_pubmed_pmc")
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```

```
{"model_id":"1ab63b25a3074a56ab6c0e293fd0e37c","version_major":2,"vers
ion_minor":0}

{"model_id":"e698d44e7b1a423fbafb129b9184ab0a","version_major":2,"vers
ion_minor":0}

{"model_id":"c09430a542a84eda935ff451e23179e7","version_major":2,"vers
ion_minor":0}

{"model_id":"b9f9ec80fd87438095b32139fbc8cae2","version_major":2,"vers
ion_minor":0}
```

```
dataset_paths = {
    "train": "/content/train.tsv",
    "test": "/content/test.tsv"
}

def read_dataset(path):
    with open(path, 'r') as f:
        raw_data = f.read().strip().split('\n\n')
    return raw_data
```

```python
datasets = {split: read_dataset(path) for split, path in
dataset_paths.items()}

def process_data(raw_data):
    sents, tags_li = [], []
    for entry in raw_data:
        words = [line.split()[0] for line in entry.splitlines()]
        tags = [line.split()[-1] for line in entry.splitlines()]
        sents.append(["[CLS]"] + words + ["[SEP]"])
        tags_li.append(["<PAD>"] + tags + ["<PAD>"])

    processed_sents = []
    processed_tags = []
    for words, tags in zip(sents, tags_li):
        token_ids, label_ids = [], []
        for word, tag in zip(words, tags):
            tokens = tokenizer.tokenize(word) if word not in ("[CLS]",
"[SEP]") else [word]
            token_ids.extend(tokenizer.convert_tokens_to_ids(tokens))
            label_ids.extend([tag2idx[tag]] + [tag2idx["<PAD>"]] *
(len(tokens) - 1))
        processed_sents.append(token_ids)
        processed_tags.append(label_ids)
    return processed_sents, processed_tags

processed_datasets = {split: process_data(data) for split, data in
datasets.items()}

train_sents, train_tags = processed_datasets["train"]
print("Sample Processed Sentence (Tokens):", train_sents[0])
print("Sample Processed Tags:", train_tags[0])
```

```
Sample Processed Sentence (Tokens): [101, 22087, 27412, 18575, 1673,
118, 10645, 2112, 12602, 177, 1183, 11439, 5026, 1988, 1107, 22195,
112, 188, 3653, 131, 170, 23191, 2025, 1113, 1103, 3154, 1104, 3850,
10602, 119, 102]
Sample Processed Tags: [0, 2, 0, 0, 0, 1, 1, 3, 0, 4, 0, 0, 0, 0, 1,
3, 4, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0]
```

```python
class NerDataset(Dataset):
    def __init__(self, sents, tags):
        self.sents = sents
        self.tags = tags

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

    def __getitem__(self, idx):
        return self.sents[idx], self.tags[idx], len(self.sents[idx])

def pad(batch):
    f = lambda x: [sample[x] for sample in batch]
    sents = f(0)
    tags = f(1)
    seqlens = f(2)
    maxlen = max(seqlens)


    pad_fn = lambda x, maxlen: [sample + [0] * (maxlen - len(sample))
for sample in x]
    padded_sents = pad_fn(sents, maxlen)
    padded_tags = pad_fn(tags, maxlen)

    return torch.LongTensor(padded_sents),
torch.LongTensor(padded_tags), torch.LongTensor(seqlens)

train_dataset = NerDataset(train_sents, train_tags)
train_loader = DataLoader(dataset=train_dataset, batch_size=32,
shuffle=True, collate_fn=pad)

test_dataset = NerDataset(*processed_datasets["test"])
test_loader = DataLoader(dataset=test_dataset, batch_size=32,
shuffle=False, collate_fn=pad)

for batch in train_loader:
    batch_sents, batch_tags, batch_seqlens = batch
    print("Batch Sentences Shape:", batch_sents.shape)
    print("Batch Tags Shape:", batch_tags.shape)
    print("Batch Sequence Lengths:", batch_seqlens.shape)
    break
```

```
Batch Sentences Shape: torch.Size([32, 93])
Batch Tags Shape: torch.Size([32, 93])
Batch Sequence Lengths: torch.Size([32])
```

```
class NERModel(nn.Module):
    def __init__(self, vocab_len, device='cpu'):
        super(NERModel, self).__init__()
        self.bert =
AutoModel.from_pretrained("monologg/biobert_v1.0_pubmed_pmc")
        self.fc = nn.Linear(768, vocab_len)
        self.device = device

    def forward(self, x, y=None):

        x = x.to(self.device)
        attention_mask = (x > 0).to(self.device)


        bert_output = self.bert(input_ids=x,
attention_mask=attention_mask)
        encoded_layers = bert_output.last_hidden_state
        logits = self.fc(encoded_layers)
        y_hat = logits.argmax(-1)

        return logits, y.to(self.device) if y is not None else None,
y_hat

device = 'cuda' if torch.cuda.is_available() else 'cpu'
model = NERModel(vocab_len=len(VOCAB), device=device).to(device)
print("Model Initialized!")
```

```
{"model_id":"e810b60afefb4055ae4092764ee788bd","version_major":2,"vers
ion_minor":0}
```

```
Model Initialized!
```

```
optimizer = optim.Adam(model.parameters(), lr=0.0001)
criterion = nn.CrossEntropyLoss(ignore_index=0)

# Training loop
def train_model(model, train_loader, optimizer, criterion, device,
n_epochs=10):
    model.train()
    for epoch in range(1, n_epochs + 1):
        print(f"Epoch {epoch}/{n_epochs}")
        total_loss = 0
        total_correct = 0
        total_tokens = 0

        for i, batch in enumerate(train_loader):
            x, y, seqlens = batch
            x, y = x.to(device), y.to(device)

            optimizer.zero_grad()
            logits, _, y_hat = model(x, y)
            logits = logits.view(-1, logits.shape[-1])
            y = y.view(-1)

            loss = criterion(logits, y)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()


            y_hat = y_hat.view(-1)
            mask = y != 0
            correct = (y_hat == y) & mask
            total_correct += correct.sum().item()
            total_tokens += mask.sum().item()
        avg_loss = total_loss / len(train_loader)
        print(f"Epoch {epoch} finished. Average Loss: {avg_loss:.4f}")

%%time
train_model(model, train_loader, optimizer, criterion, device)
```

```
Epoch 1/10
Epoch 1 finished. Average Loss: 0.1390
Epoch 2/10
Epoch 2 finished. Average Loss: 0.0386
Epoch 3/10
Epoch 3 finished. Average Loss: 0.0187
Epoch 4/10
Epoch 4 finished. Average Loss: 0.0195
Epoch 5/10
Epoch 5 finished. Average Loss: 0.0167
Epoch 6/10
Epoch 6 finished. Average Loss: 0.0104
Epoch 7/10
Epoch 7 finished. Average Loss: 0.0078
Epoch 8/10
Epoch 8 finished. Average Loss: 0.0071
Epoch 9/10
Epoch 9 finished. Average Loss: 0.0086
Epoch 10/10
Epoch 10 finished. Average Loss: 0.0048
CPU times: user 11min 59s, sys: 1.08 s, total: 12min
Wall time: 12min 4s
```

```
from sklearn.metrics import classification_report

def test_model_fixed(model, test_loader, device):
    model.eval()
    all_preds = []
    all_labels = []
    all_words = []

    with torch.no_grad():
        for batch in test_loader:
            x, y, seqlens = batch
            x, y = x.to(device), y.to(device)


            logits, _, y_hat = model(x)
            y_hat = y_hat.cpu().numpy()
            y = y.cpu().numpy()


            for i in range(len(seqlens)):
                seq_len = seqlens[i]
                preds = y_hat[i][:seq_len]
                labels = y[i][:seq_len]
                all_preds.extend(preds)
                all_labels.extend(labels)


    flat_preds = [p for p, l in zip(all_preds, all_labels) if l != 0]
    flat_labels = [l for l in all_labels if l != 0]

    valid_labels = VOCAB[2:]
    valid_indices = list(range(2, len(VOCAB)))

    print("Classification Report:")
    print(classification_report(flat_labels, flat_preds,
target_names=valid_labels, labels=valid_indices))


    return all_preds, all_labels

predictions, true_labels = test_model_fixed(model, test_loader,
device)
```

```
Classification Report:
              precision    recall  f1-score   support

  B-Chemical       0.93      0.93      0.93      5385
   B-Disease       0.83      0.86      0.84      4424
   I-Disease       0.76      0.81      0.78      2737
  I-Chemical       0.80      0.88      0.84      1628

   micro avg       0.85      0.88      0.86     14174
   macro avg       0.83      0.87      0.85     14174
weighted avg       0.85      0.88      0.86     14174
```

```
def predict_sentence(model, sentence, tokenizer, tag2idx, idx2tag,
device):
    model.eval()


    tokens = tokenizer.tokenize(sentence)
    input_ids = tokenizer.convert_tokens_to_ids(["[CLS]"] + tokens +
["[SEP]"])
    input_tensor = torch.tensor(input_ids).unsqueeze(0).to(device)


    with torch.no_grad():
        logits, _, y_hat = model(input_tensor)
        y_hat = y_hat.squeeze(0).cpu().numpy()


    predicted_tags = [idx2tag[idx] for idx in y_hat[1:-1]]
    token_tag_pairs = list(zip(tokens, predicted_tags))
    return token_tag_pairs


input_sentence = "Selegiline induced postural hypotension in
Parkinson's disease."

predicted_tags = predict_sentence(model, input_sentence, tokenizer,
tag2idx, {v: k for k, v in tag2idx.items()}, device)


print("Predictions:")
for token, tag in predicted_tags:
    print(f"{token}\t{tag}")
```

```
Predictions:
Se    B-Chemical
##leg B-Chemical
##ili I-Chemical
##ne  I-Chemical
induced     0
post  B-Disease
##ural     I-Disease
h     I-Disease
##y   I-Disease
##pot I-Disease
##ens I-Disease
##ion I-Disease
in    0
Parkinson  B-Disease
'     I-Disease
s     I-Disease
disease    I-Disease
.     0
```