```
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":"27d3f689d2be4bc1a42149e114acb7b7","version_major":2,"vers
ion_minor":0}

{"model_id":"9bd6112641e5403296ec461daba22be0","version_major":2,"vers
ion_minor":0}

{"model_id":"c1a60b793419482e811a0ef5d32f0050","version_major":2,"vers
ion_minor":0}

{"model_id":"350083e186594414bd2b02d22d4c7faa","version_major":2,"vers
ion_minor":0}
```


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

```
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(["O"] + tags + ["O"])

    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)
            token_ids.extend(tokenizer.convert_tokens_to_ids(tokens))

            label_ids.extend([tag2idx[tag]] * len(tokens))
        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: [1, 2, 2, 2, 2, 1, 1, 3, 3, 4, 4, 4, 4, 4, 1,
3, 4, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
!pip install pytorch-crf
```

```
Collecting pytorch-crf
  Downloading pytorch_crf-0.7.2-py3-none-any.whl.metadata (2.4 kB)
Downloading pytorch_crf-0.7.2-py3-none-any.whl (9.5 kB)
Installing collected packages: pytorch-crf
Successfully installed pytorch-crf-0.7.2
```

```
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)

    # Padding
    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)

devel_dataset = NerDataset(*processed_datasets["devel"])
devel_loader = DataLoader(dataset=devel_dataset, batch_size=32,
shuffle=False, 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, 79])
Batch Tags Shape: torch.Size([32, 79])
Batch Sequence Lengths: torch.Size([32])

import torch
import torch.nn as nn
from transformers import AutoModel
from torchcrf import CRF
```

```
class NERModelWithCRF(nn.Module):
    def __init__(self, vocab_len, device='cpu'):
        super(NERModelWithCRF, self).__init__()
        self.device = device


        self.bert =
AutoModel.from_pretrained("monologg/biobert_v1.0_pubmed_pmc")


        self.fc = nn.Linear(self.bert.config.hidden_size, vocab_len)

        # CRF layer
        self.crf = CRF(vocab_len, batch_first=True)

    def forward(self, x, tags=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  #


        emissions = self.fc(encoded_layers)

        if tags is not None:

            log_likelihood = self.crf(emissions, tags,
mask=attention_mask)
            return -log_likelihood
        else:

            predictions = self.crf.decode(emissions,
mask=attention_mask)
            return predictions

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

{"model_id":"ac5376df038e4e76bd1fc1d850579499","version_major":2,"version_minor":0}

```
Model Initialized!
```

```
optimizer = optim.Adam(model.parameters(), lr=1e-5)
```

```
criterion = nn.CrossEntropyLoss()


def train_model_with_crf(model, train_loader, optimizer, device,
n_epochs=3):
    model.train()
    for epoch in range(1, n_epochs + 1):
        print(f"Epoch {epoch}/{n_epochs}")
        total_loss = 0

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

            optimizer.zero_grad()

            # Compute loss using CRF
            loss = model(x, y)
            loss.backward()
            optimizer.step()

            total_loss += loss.item()

        avg_loss = total_loss / len(train_loader)
        print(f"Epoch {epoch} finished. Average Loss: {avg_loss:.4f}")

%%time
train_model_with_crf(model, train_loader, optimizer, device,
n_epochs=10)
```

```
Epoch 1/10
Epoch 1 finished. Average Loss: 559.2905
Epoch 2/10
Epoch 2 finished. Average Loss: 174.7533
Epoch 3/10
Epoch 3 finished. Average Loss: 106.6361
Epoch 4/10
Epoch 4 finished. Average Loss: 74.7410
Epoch 5/10
Epoch 5 finished. Average Loss: 53.5788
Epoch 6/10
Epoch 6 finished. Average Loss: 38.9703
Epoch 7/10
Epoch 7 finished. Average Loss: 28.6358
Epoch 8/10
Epoch 8 finished. Average Loss: 21.9491
Epoch 9/10
Epoch 9 finished. Average Loss: 17.6526
Epoch 10/10
Epoch 10 finished. Average Loss: 13.7372
```

```
CPU times: user 13min 20s, sys: 1.97 s, total: 13min 22s
Wall time: 13min 53s
```

```
from sklearn.metrics import classification_report

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

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


            predictions = model(x)
            for i, seq_len in enumerate(seqlens):
                all_preds.extend(predictions[i][:seq_len])
                all_labels.extend(y[i][:seq_len].cpu().numpy())  \


    flat_preds = [p for p, l in zip(all_preds, all_labels) if l !=
tag2idx["<PAD>"]]
    flat_labels = [l for l in all_labels if l != tag2idx["<PAD>"]]



    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 flat_preds, flat_labels

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

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

  B-Chemical       0.96      0.95      0.95     17967
   B-Disease       0.87      0.88      0.87     12489
   I-Disease       0.76      0.82      0.79      5338
  I-Chemical       0.83      0.90      0.87      2921

   micro avg       0.89      0.91      0.90     38715
   macro avg       0.85      0.89      0.87     38715
weighted avg       0.89      0.91      0.90     38715
```

```
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():
        predictions = model(input_tensor)


    predicted_tags = [idx2tag[idx] for idx in predictions[0][1:-1]]


    token_tag_pairs = list(zip(tokens, predicted_tags))
    return token_tag_pairs

e
input_sentence = "i was diagonised with malaria,chickenpox and
smallpox"


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:
i     O
was   O
di    O
##agon      O
##ised      O
with  O
malaria     B-Disease
,     O
chicken     B-Disease
##pox B-Disease
and   O
small B-Disease
##pox B-Disease
```