```
import pandas as pd
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from torch.nn.utils.rnn import pad_sequence
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

```
Using device: cpu
```

```
# Load and prepare data
data = pd.read_csv("ner_dataset.csv", encoding="latin1").ffill()
words = list(data["Word"].unique())
tags = list(data["Tag"].unique())

if "ENDPAD" not in words:
    words.append("ENDPAD")

word2idx = {w: i + 1 for i, w in enumerate(words)}
tag2idx = {t: i for i, t in enumerate(tags)}
idx2tag = {i: t for t, i in tag2idx.items()}
```

```
data.head(50)
```

	Sentence #	Word	POS	Tag
0	Sentence: 1	Thousands	NNS	O
1	Sentence: 1	of	IN	O
2	Sentence: 1	demonstrators	NNS	O
3	Sentence: 1	have	VBP	O
4	Sentence: 1	marched	VBN	O
5	Sentence: 1	through	IN	O
6	Sentence: 1	London	NNP	B-geo
7	Sentence: 1	to	TO	O
8	Sentence: 1	protest	VB	O
9	Sentence: 1	the	DT	O
10	Sentence: 1	war	NN	O
11	Sentence: 1	in	IN	O
12	Sentence: 1	Iraq	NNP	B-geo
13	Sentence: 1	and	CC	O
14	Sentence: 1	demand	VB	O
15	Sentence: 1	the	DT	O
16	Sentence: 1	withdrawal	NN	O
17	Sentence: 1	of	IN	O
18	Sentence: 1	British	JJ	B-gpe
19	Sentence: 1	troops	NNS	O
20	Sentence: 1	from	IN	O
22	Sentence: 1	country	NN	O
23	Sentence: 1	.	.	O
24	Sentence: 2	Families	NNS	O
26	Sentence: 2	soldiers	NNS	O
27	Sentence: 2	killed	VBN	O

Essential info about tagged entities:

geo = Geographical Entity org = Organization per = Person gpe = Geopolitical Entity tim = Time indicator art = Artifact eve = Event nat = Natural Phenomenon

```
print("Unique words in corpus:", data['Word'].nunique())
print("Unique tags in corpus:", data['Tag'].nunique())
```

```
Unique words in corpus: 18758
Unique tags in corpus: 17
```

```
print("Unique tags are:", tags)
```

```
Unique tags are: ['O', 'B-geo', 'B-gpe', 'B-per', 'I-geo', 'B-org', 'I-org', 'B-
```

```
# Group words by sentences
class SentenceGetter:
    def __init__(self, data):
        self.grouped = data.groupby("Sentence #", group_keys=False).apply(
            lambda s: [(w, t) for w, t in zip(s["Word"], s["Tag"])]
        )
        self.sentences = list(self.grouped)
```

```
getter = SentenceGetter(data)
sentences = getter.sentences
```

34	Sentence: 2	who	WP	O
35	Sentence: 2	banners	NNS	O
36	Sentence: 2	with	IN	O
37	Sentence: 2	such	JJ	O
38	Sentence: 2	slogans	NNS	O
39	Sentence: 2	as	IN	O
40	Sentence: 2	"	``	O
41	Sentence: 2	Bush	NNP	B-per
42	Sentence: 2	Number	NN	O
43	Sentence: 2	One	CD	O
44	Sentence: 2	Terrorist	NN	O
45	Sentence: 2	"	``	O
46	Sentence: 2	and	CC	O
47	Sentence: 2	"	``	O
48	Sentence: 2	Stop	VB	O
49	Sentence: 2			

```
sentences[35]
```

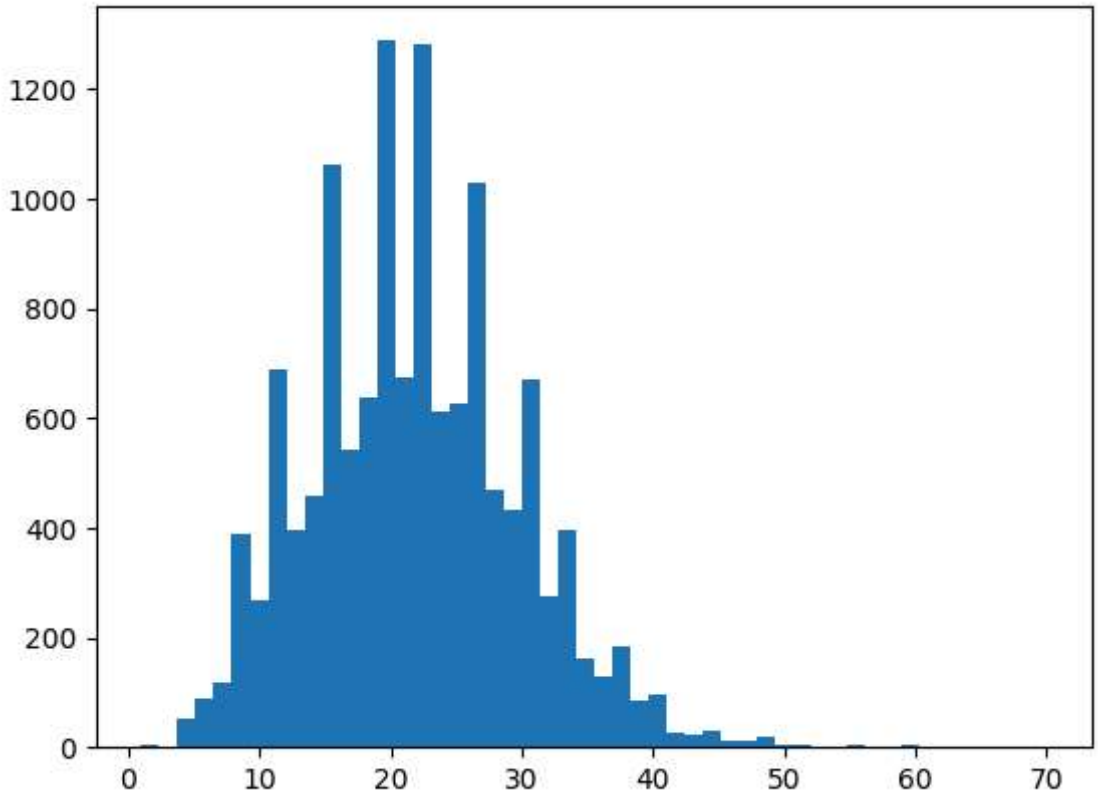
```
[('The', 'O'),
 ('U.S.', 'B-org'),
 ('Geological', 'I-org'),
 ('Survey', 'I-org'),
 ('gave', 'O'),
 ('a', 'O'),
 ('preliminary', 'O'),
 ('estimate', 'O'),
 ('of', 'O'),
 ('the', 'O'),
 ('strength', 'O'),
 ('of', 'O'),
 ('the', 'O'),
 ('Tuesday', 'B-tim'),
 ('morning', 'I-tim'),
 ('quake', 'O'),
 ('at', 'O'),
 ('6.7', 'O'),
 ('on', 'O'),
 ('the', 'O'),
 ('Richter', 'B-geo'),
 ('scale', 'O'),
 ('', 'O'),
 ('and', 'O'),
 ('said', 'O'),
 ('the', 'O'),
 ('epicenter', 'O'),
 ('was', 'O'),
 ('close', 'O'),
 ('to', 'O'),
 ('the', 'O'),
 ('island', 'O'),
 ('of', 'O'),
 ('Nias', 'B-org'),
 ('.', 'O')]
```

```
# Encode sentences
X = [[word2idx[w] for w, t in s] for s in sentences]
y = [[tag2idx[t] for w, t in s] for s in sentences]
```

```
word2idx
```

```
 'hunt': 953,
 'So': 954,
 'far': 955,
 'nearly': 956,
 '100': 957,
 'reported': 958,
 'On': 959,
 'dozens': 960,
 'stormed': 961,
 'checkpoint': 962,
 'At': 963,
 '32': 964,
 'counter-attack': 965,
 'Elsewhere': 966,
 'found': 967,
 'bodies': 968,
 'had': 969,
 'Kurram': 970,
 'along': 971,
 'Afghan': 972,
 'border': 973,
 'few': 974,
 'days': 975,
 'ago': 976,
 'separate': 977,
 'clashes': 978,
 '13': 979,
 'guerrillas': 980,
 'two': 981,
 'encounters': 982,
 'central': 983,
 'Uruzgan': 984,
 'others': 985,
 'injured': 986,
 'fighting': 987,
 'Another': 988,
 'eastern': 989,
 'Paktika': 990,
 'Separately': 991,
 'NATO-led': 992,
 'peacekeeping': 993,
 'mission': 994,
 'early': 995,
 'Mazar-e-Sharif': 996,
 'motivated': 997,
 'spared': 998,
 'bloodshed': 999,
 'plagued': 1000,
 ...}
```

```
plt.hist([len(s) for s in sentences], bins=50)
plt.show()
```




```
# Pad sequences
max_len = 50
X_pad = pad_sequence([torch.tensor(seq) for seq in X], batch_first=True, paddir
y_pad = pad_sequence([torch.tensor(seq) for seq in y], batch_first=True, paddir
X_pad = X_pad[:, :max_len]
y_pad = y_pad[:, :max_len]
```

```
X_pad[0]
```

```
tensor([    1,     2,     3,     4,     5,     6,     7,     8,     9,    10,
           11,    12,    13,    14,    15,    10,    16,     2,    17,    18,
           19,    20,    21,    22, 18759, 18759, 18759, 18759, 18759, 18759,
        18759, 18759, 18759, 18759, 18759, 18759, 18759, 18759, 18759, 18759,
        18759, 18759, 18759, 18759, 18759, 18759, 18759, 18759, 18759, 18759])
```

```
y_pad[0]
```

```
tensor([0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0])
```

```
# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X_pad, y_pad, test_size=0.2
```

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

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

    def __getitem__(self, idx):
        return {
            "input_ids": self.X[idx],
            "labels": self.y[idx]
        }

train_loader = DataLoader(NERDataset(X_train, y_train), batch_size=32, shuffle=
test_loader = DataLoader(NERDataset(X_test, y_test), batch_size=32)
```

```
class BiLSTMTagger(nn.Module):
  def __init__(self, vocab_size, tagset_size, embedding_dim = 50, hidden_dim =
    super(BiLSTMTagger, self).__init__()
    self.embedding = nn.Embedding(vocab_size, embedding_dim)
    self.dropout = nn.Dropout(0,1)
    self.lstm = nn.LSTM(embedding_dim, hidden_dim, bidirectional=True, batch_fi
    self.fc = nn.Linear(hidden_dim * 2, tagset_size)

  def forward(self, x):
    x = self.embedding(x)
    x = self.dropout(x)
    x, _ = self.lstm(x)
    return self.fc(x)
```

```
model = BiLSTMTagger(len(word2idx)+1, len(tag2idx)).to(device)
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

```
# Training and Evaluation Functions
def train_model(model, train_loader, test_loader, loss_fn, optimizer, epochs=3)
    train_losses, val_losses = [], []
    for epoch in range(epochs):
      model.train()
      total_loss = 0
      for batch in train_loader:
        input_ids = batch["input_ids"].to(device)
        labels = batch["labels"].to(device)
        optimizer.zero_grad()
        outputs = model(input_ids)
        loss = loss_fn(outputs.view(-1, len(tag2idx)), labels.view(-1))
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
      train_losses.append(total_loss)

      model.eval()
      val_loss = 0
      with torch.no_grad():
        for batch in test_loader:
          input_ids = batch["input_ids"].to(device)
```

```
            labels = batch['labels'].to(device)
            outputs = model(input_ids)
            loss = loss_fn(outputs.view(-1, len(tag2idx)), labels.view(-1))
            val_loss += loss.item()
        val_losses.append(val_loss)
        print(f"Epoch {epoch+1}: Train Loss = {total_loss:.4f}, Val Loss = {val_l

    return train_losses, val_losses
```

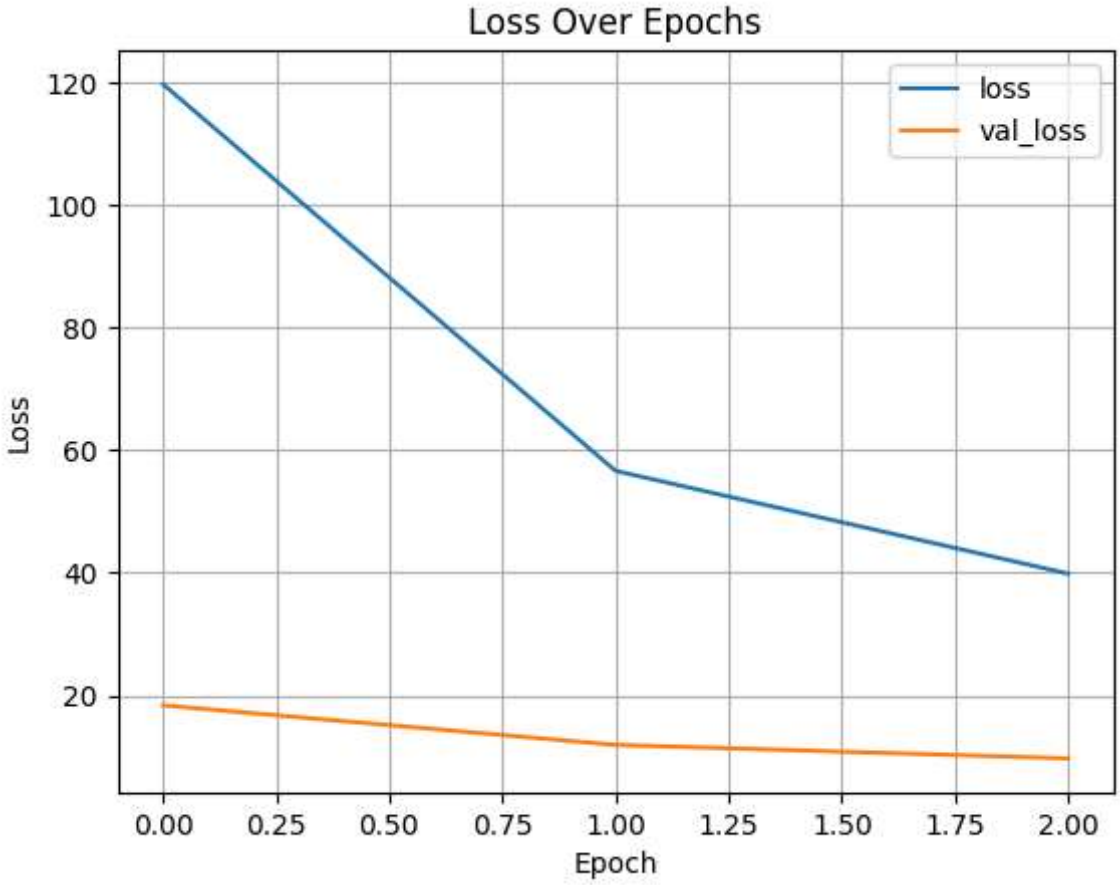
```
def evaluate_model(model, test_loader, X_test, y_test):
    model.eval()
    true_tags, pred_tags = [], []
    with torch.no_grad():
        for batch in test_loader:
            input_ids = batch["input_ids"].to(device)
            labels = batch["labels"].to(device)
            outputs = model(input_ids)
            preds = torch.argmax(outputs, dim=-1)
            for i in range(len(labels)):
                for j in range(len(labels[i])):
                    if labels[i][j] != tag2idx["O"]:
                        true_tags.append(idx2tag[labels[i][j].item()])
                        pred_tags.append(idx2tag[preds[i][j].item()])
```

```
# Run training and evaluation
train_losses, val_losses = train_model(model, train_loader, test_loader, loss_f
evaluate_model(model, test_loader, X_test, y_test)
```

```
Epoch 1: Train Loss = 119.6465, Val Loss = 18.3950
Epoch 2: Train Loss = 56.6193, Val Loss = 11.9464
Epoch 3: Train Loss = 39.8621, Val Loss = 9.7485
```

```
# Plot loss
print('Name: A.LAHARI')
print('Register Number: 212223230111')
history_df = pd.DataFrame({"loss": train_losses, "val_loss": val_losses})
history_df.plot(title="Loss Over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
```

```
Name: A.LAHARI
Register Number: 212223230111
```




```
# Inference and prediction
i = 125
model.eval()
sample = X_test[i].unsqueeze(0).to(device)
output = model(sample)
preds = torch.argmax(output, dim=-1).squeeze().cpu().numpy()
true = y_test[i].numpy()

print('Name: A.LAHARI')
print('Register Number: 212223230111')
print("{:<15} {:<10} {}\n{}".format("Word", "True", "Pred", "-" * 40))
for w_id, true_tag, pred_tag in zip(X_test[i], y_test[i], preds):
    if w_id.item() != word2idx["ENDPAD"]:
        word = words[w_id.item() - 1]
        true_label = tags[true_tag.item()]
        pred_label = tags[pred_tag]
        print(f"{word:<15} {true_label:<10} {pred_label}")
```

```
Name: A.LAHARI
Register Number: 212223230111
Word            True       Pred
----------------------------------------
In              O          O
a               O          O
poll            O          O
of              O          O
1,000           O          O
adults          O          O
```

```
taken           O          O
October         B-tim      B-tim
3               I-tim      O
through         I-tim      O
5               I-tim      O
,               O          O
the             O          O
```