# HW4

November 10, 2023

[828]:
```
import pandas as pd
import numpy as np
import datasets
import torch
import math
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
from sklearn.metrics import precision_score, recall_score, f1_score,␣
 ↪classification_report
from torch.utils.data import TensorDataset
import time
from itertools import chain
from torch.nn.utils.rnn import pad_sequence
import torch.nn.functional as F
import copy
from torch.optim import lr_scheduler
```

[48]:
```
import torch
import math
# this ensures that the current MacOS version is at least 12.3+
print(torch.backends.mps.is_available())
# this ensures that the current current PyTorch installation was built with MPS␣
 ↪activated.
print(torch.backends.mps.is_built())
```

```
True
True
```

[49]:
```
dtype = torch.float
device = torch.device("mps")
```

[244]:
```
dataset = datasets.load_dataset("conll2003")
```

### 0.0.1 Convert words/tokens to indices

[683]:
```
import itertools
from collections import Counter

word_frequency = Counter(itertools.chain(*dataset['train']['tokens']))  # type:␣
↪ignore

# Remove words below threshold 3
word_frequency = {
    word: frequency
    for word, frequency in word_frequency.items()
    if frequency >= 3
}

word2idx = {
    word: index
    for index, word in enumerate(word_frequency.keys(), start=2)
}

word2idx['[PAD]'] = 0
word2idx['[UNK]'] = 1
```

[1004]:
```
sample_tokens = dataset['train'][0]['tokens']
sample_tokens
```

[1004]:
```
['EU', 'rejects', 'German', 'call', 'to', 'boycott', 'British', 'lamb', '.']
```

[684]:
```
# the vocab size
vocab_size = max(word2idx.values())+1
vocab_size
```

[684]:
```
8128
```

[689]:
```
def convert_word_to_id(sample):
#Code to convert all tokens to their respective indexes
#If the token is unknown, we set index of 1
    input_ids = [ word2idx.get(token, 1) for token in sample['tokens'] ]

    sample['input_ids'] = input_ids
    return sample

dataset = dataset.map(convert_word_to_id)
```

```
Map:   0%|          | 0/14041 [00:00<?, ? examples/s]

Map:   0%|          | 0/3250 [00:00<?, ? examples/s]

Map:   0%|          | 0/3453 [00:00<?, ? examples/s]
```

[692]:
```python
df_train = pd.DataFrame(dataset['train']).drop(columns=['pos_tags',␣
↪'chunk_tags', 'id', 'tokens'])
df_train.columns = ['label','input_ids']

df_test = pd.DataFrame(dataset['test']).drop(columns=['pos_tags', 'chunk_tags',␣
↪'id', 'tokens'])
df_test.columns = ['label','input_ids']

df_val = pd.DataFrame(dataset['validation']).drop(columns=['pos_tags',␣
↪'chunk_tags', 'id', 'tokens'])
df_val.columns = ['label','input_ids']
```

### 0.0.2 Padding

[1145]:
```python
import pandas as pd
import torch
from torch.utils.data import Dataset

# Create a custom Dataset class
class CustomDataset(Dataset):
    def __init__(self, dataframe):
        self.data = dataframe

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

    def __getitem__(self, idx):
        label = torch.tensor(self.data.loc[idx, "label"], dtype=torch.long)
        input_ids = torch.tensor(self.data.loc[idx, "input_ids"], dtype=torch.
↪long)

        return input_ids, label

# Create an instance of the CustomDataset
dataset_train = CustomDataset(df_train)

# Example: Accessing a single sample
print(dataset_train[2])
```

```
(tensor([12, 13]), tensor([5, 0]))
```

[1146]:
```python
def custom_collate(batch):
    # Separate input sequences and labels
    input_seqs, labels = zip(*batch)

    # Calculate the sequence lengths based on input sequences (assuming they␣
↪have the same length as labels)
```

```
    sequence_lengths = [len(seq) for seq in input_seqs]

    # Sort input sequences and labels by sequence length (descending)
    sorted_seqs_and_labels = sorted(zip(input_seqs, labels), key=lambda x:␣
↪len(x[0]), reverse=True)
    sorted_input_seqs, sorted_labels = zip(*sorted_seqs_and_labels)

    # Pad input sequences to the maximum length within the batch
    padded_input_seqs = pad_sequence(sorted_input_seqs, batch_first=True,␣
↪padding_value=0)  # Use 0 as the padding value
    padded_labels = pad_sequence(sorted_labels, batch_first=True,␣
↪padding_value=0)  # Use 0 as the padding value

    return padded_input_seqs, padded_labels
```

### 0.0.3 Create dataloaders

[1147]:
```
def dataloader_generator(df,shuffle):
    dataset_from_df = CustomDataset(df)
    batch_size = 64
    dataloader = DataLoader(dataset_from_df, batch_size=batch_size,␣
↪collate_fn=custom_collate, shuffle=shuffle)
    return dataloader

train_loader  = dataloader_generator(df_train,shuffle=True)
test_loader  = dataloader_generator(df_test,shuffle=False)
val_loader  = dataloader_generator(df_val,shuffle=False)

for batch in val_loader:
    input_val, target_val = batch
    break
```

### 0.0.4 Building the model

[340]:
```
!wget https://raw.githubusercontent.com/sighsmile/conlleval/master/conlleval.py
```

```
--2023-11-06 16:54:57--
https://raw.githubusercontent.com/sighsmile/conlleval/master/conlleval.py
    raw.githubusercontent.com (raw.githubusercontent.com)… 185.199.108.133,
185.199.109.133, 185.199.110.133, …
   raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443…
  HTTP      … 200 OK
 7502 (7.3K) [text/plain]
  : "conlleval.py.1"

conlleval.py.1      100%[===================>]   7.33K  --.-KB/s    0s
```

```
2023-11-06 16:54:57 (14.7 MB/s) -   "conlleval.py.1" [7502/7502])
```

[341]:
```python
from conlleval import evaluate
```

[1150]:
```python
class BiLSTMNER(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim,␣
 ↪num_layers, dropout):
        super(BiLSTMNER, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.bilstm = nn.LSTM(embedding_dim, hidden_dim, num_layers=num_layers,
                              batch_first=True, bidirectional=True)
        self.dropout = nn.Dropout(dropout)
        self.linear = nn.Linear(hidden_dim * 2, output_dim)
        self.elu = nn.ELU()
        self.classifier = nn.Linear(output_dim, num_tags)  # num_tags is the␣
 ↪number of unique NER tags

    def forward(self, x):
        x = self.embedding(x)
        x, _ = self.bilstm(x)
        x = self.dropout(x)
        x = self.linear(x)
        x = self.elu(x)
        x = self.classifier(x)
        return x

#initialize
num_tags = 9
vocab_size = max(word2idx.values())+1

model = BiLSTMNER(vocab_size, 100, 256, 128, 1, 0.33)
optimizer = optim.Adam(model.parameters(), lr=0.001)
loss_function = nn.CrossEntropyLoss()

#training
num_epochs = 20
print('start training')
for epoch in range(num_epochs):
    start_time = time.time()
    model.train()
    total_loss = 0
    for batch in train_loader:
        optimizer.zero_grad()
        inputs, targets = batch
        outputs = model(inputs)
```

```
        batch_size = inputs.size()[-1]
        #From the instruction of CrossEntropy, we need to change the format of␣
↪outputs
        loss = loss_function(outputs.permute(0,2,1), targets)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    end_time = time.time()
    print(f'Epoch {epoch + 1}, Loss: {total_loss / len(train_loader)}, time:␣
↪{end_time-start_time}s')
    print('validation error: ')
    precision, recall, f1 = eval(model, val_loader)
```

```
start training
Epoch 1, Loss: 0.24208735396916217, time: 52.45967507362366s
validation error:
processed 152266 tokens with 5942 phrases; found: 2102 phrases; correct: 1080.
accuracy:  20.82%; (non-O)
accuracy:  95.34%; precision:  51.38%; recall:  18.18%; FB1:  26.85
              LOC: precision:  59.36%; recall:  26.08%; FB1:  36.23  807
             MISC: precision:  33.33%; recall:   0.33%; FB1:   0.64  9
              ORG: precision:  30.00%; recall:   3.36%; FB1:   6.04  150
              PER: precision:  48.68%; recall:  30.02%; FB1:  37.14  1136
Epoch 2, Loss: 0.11488994293930856, time: 47.83418798446655s
validation error:
processed 152266 tokens with 5942 phrases; found: 4661 phrases; correct: 2929.
accuracy:  52.38%; (non-O)
accuracy:  97.05%; precision:  62.84%; recall:  49.29%; FB1:  55.25
              LOC: precision:  72.27%; recall:  65.11%; FB1:  68.50  1655
             MISC: precision:  62.36%; recall:  37.20%; FB1:  46.60  550
              ORG: precision:  45.43%; recall:  44.44%; FB1:  44.93  1312
              PER: precision:  69.41%; recall:  43.11%; FB1:  53.18  1144
Epoch 3, Loss: 0.06882393922318111, time: 48.614689111709595s
validation error:
processed 152266 tokens with 5942 phrases; found: 5102 phrases; correct: 3706.
accuracy:  66.22%; (non-O)
accuracy:  97.86%; precision:  72.64%; recall:  62.37%; FB1:  67.11
              LOC: precision:  85.42%; recall:  69.84%; FB1:  76.85  1502
             MISC: precision:  65.60%; recall:  57.70%; FB1:  61.40  811
              ORG: precision:  62.67%; recall:  54.21%; FB1:  58.14  1160
              PER: precision:  71.45%; recall:  63.19%; FB1:  67.07  1629
Epoch 4, Loss: 0.04578833337026564, time: 51.33672094345093s
validation error:
processed 152266 tokens with 5942 phrases; found: 5280 phrases; correct: 4048.
accuracy:  71.87%; (non-O)
accuracy:  98.20%; precision:  76.67%; recall:  68.13%; FB1:  72.14
              LOC: precision:  89.86%; recall:  74.80%; FB1:  81.64  1529
```

```
           MISC: precision:  75.29%; recall:  63.45%; FB1:  68.86  777
            ORG: precision:  62.88%; recall:  63.16%; FB1:  63.02  1347
            PER: precision:  76.34%; recall:  67.43%; FB1:  71.61  1627
Epoch 5, Loss: 0.032878815653649245, time: 48.04787993431091s
validation error:
processed 152266 tokens with 5942 phrases; found: 5393 phrases; correct: 4290.
accuracy:  75.31%; (non-O)
accuracy:  98.40%; precision:  79.55%; recall:  72.20%; FB1:  75.69
            LOC: precision:  85.29%; recall:  80.19%; FB1:  82.66  1727
           MISC: precision:  80.28%; recall:  68.87%; FB1:  74.14  791
            ORG: precision:  73.72%; recall:  64.21%; FB1:  68.63  1168
            PER: precision:  77.39%; recall:  71.72%; FB1:  74.44  1707
Epoch 6, Loss: 0.023866635279475964, time: 48.24751901626587s
validation error:
processed 152266 tokens with 5942 phrases; found: 5465 phrases; correct: 4370.
accuracy:  76.53%; (non-O)
accuracy:  98.46%; precision:  79.96%; recall:  73.54%; FB1:  76.62
            LOC: precision:  88.65%; recall:  79.91%; FB1:  84.05  1656
           MISC: precision:  80.98%; recall:  69.74%; FB1:  74.94  794
            ORG: precision:  71.36%; recall:  67.26%; FB1:  69.25  1264
            PER: precision:  77.50%; recall:  73.67%; FB1:  75.54  1751
Epoch 7, Loss: 0.017713668648238208, time: 48.44618272781372s
validation error:
processed 152266 tokens with 5942 phrases; found: 6117 phrases; correct: 4592.
accuracy:  80.59%; (non-O)
accuracy:  98.36%; precision:  75.07%; recall:  77.28%; FB1:  76.16
            LOC: precision:  85.71%; recall:  82.63%; FB1:  84.15  1771
           MISC: precision:  78.41%; recall:  70.50%; FB1:  74.24  829
            ORG: precision:  62.20%; recall:  70.92%; FB1:  66.27  1529
            PER: precision:  74.09%; recall:  79.97%; FB1:  76.92  1988
Epoch 8, Loss: 0.013745928631926125, time: 47.203505992889404s
validation error:
processed 152266 tokens with 5942 phrases; found: 5289 phrases; correct: 4409.
accuracy:  76.44%; (non-O)
accuracy:  98.53%; precision:  83.36%; recall:  74.20%; FB1:  78.51
            LOC: precision:  91.49%; recall:  80.78%; FB1:  85.81  1622
           MISC: precision:  80.94%; recall:  72.78%; FB1:  76.64  829
            ORG: precision:  78.06%; recall:  66.07%; FB1:  71.57  1135
            PER: precision:  80.33%; recall:  74.27%; FB1:  77.18  1703
Epoch 9, Loss: 0.010567720067179338, time: 47.350847005844116s
validation error:
processed 152266 tokens with 5942 phrases; found: 5375 phrases; correct: 4429.
accuracy:  76.74%; (non-O)
accuracy:  98.51%; precision:  82.40%; recall:  74.54%; FB1:  78.27
            LOC: precision:  91.60%; recall:  81.27%; FB1:  86.13  1630
           MISC: precision:  80.24%; recall:  73.54%; FB1:  76.74  845
            ORG: precision:  71.33%; recall:  69.20%; FB1:  70.25  1301
            PER: precision:  83.18%; recall:  72.20%; FB1:  77.30  1599
```

```
Epoch 10, Loss: 0.00821540692069737, time: 46.872527837753296s
validation error:
processed 152266 tokens with 5942 phrases; found: 5612 phrases; correct: 4468.
accuracy:  77.35%; (non-O)
accuracy:  98.49%; precision:  79.62%; recall:  75.19%; FB1:  77.34
              LOC: precision:  84.22%; recall:  84.81%; FB1:  84.51  1850
             MISC: precision:  81.68%; recall:  71.58%; FB1:  76.30  808
              ORG: precision:  71.56%; recall:  67.56%; FB1:  69.51  1266
              PER: precision:  79.62%; recall:  72.96%; FB1:  76.15  1688
Epoch 11, Loss: 0.006528246736640788, time: 46.133893966674805s
validation error:
processed 152266 tokens with 5942 phrases; found: 6042 phrases; correct: 4622.
accuracy:  80.33%; (non-O)
accuracy:  98.43%; precision:  76.50%; recall:  77.79%; FB1:  77.14
              LOC: precision:  86.12%; recall:  83.78%; FB1:  84.93  1787
             MISC: precision:  71.69%; recall:  75.27%; FB1:  73.44  968
              ORG: precision:  71.24%; recall:  68.53%; FB1:  69.86  1290
              PER: precision:  73.61%; recall:  79.80%; FB1:  76.58  1997
Epoch 12, Loss: 0.00525556694959629, time: 48.15459370613098s
validation error:
processed 152266 tokens with 5942 phrases; found: 5796 phrases; correct: 4560.
accuracy:  79.07%; (non-O)
accuracy:  98.49%; precision:  78.67%; recall:  76.74%; FB1:  77.70
              LOC: precision:  86.47%; recall:  83.12%; FB1:  84.76  1766
             MISC: precision:  79.93%; recall:  73.43%; FB1:  76.54  847
              ORG: precision:  68.11%; recall:  70.40%; FB1:  69.23  1386
              PER: precision:  78.58%; recall:  76.66%; FB1:  77.60  1797
Epoch 13, Loss: 0.00421205094052394, time: 47.617199182510376s
validation error:
processed 152266 tokens with 5942 phrases; found: 5846 phrases; correct: 4574.
accuracy:  79.32%; (non-O)
accuracy:  98.51%; precision:  78.24%; recall:  76.98%; FB1:  77.60
              LOC: precision:  86.80%; recall:  83.02%; FB1:  84.86  1757
             MISC: precision:  80.05%; recall:  73.54%; FB1:  76.65  847
              ORG: precision:  67.91%; recall:  70.40%; FB1:  69.13  1390
              PER: precision:  77.05%; recall:  77.47%; FB1:  77.26  1852
Epoch 14, Loss: 0.0038494242876450616, time: 47.95925307273865s
validation error:
processed 152266 tokens with 5942 phrases; found: 5703 phrases; correct: 4530.
accuracy:  78.86%; (non-O)
accuracy:  98.53%; precision:  79.43%; recall:  76.24%; FB1:  77.80
              LOC: precision:  87.94%; recall:  82.96%; FB1:  85.38  1733
             MISC: precision:  77.65%; recall:  75.38%; FB1:  76.50  895
              ORG: precision:  73.76%; recall:  67.71%; FB1:  70.61  1231
              PER: precision:  76.08%; recall:  76.17%; FB1:  76.13  1844
Epoch 15, Loss: 0.0037216038500032895, time: 49.66430187225342s
validation error:
processed 152266 tokens with 5942 phrases; found: 5539 phrases; correct: 4432.
```

```
accuracy:  77.23%; (non-O)
accuracy:  98.50%; precision:  80.01%; recall:  74.59%; FB1:  77.21
              LOC: precision:  88.67%; recall:  82.25%; FB1:  85.34  1704
             MISC: precision:  73.08%; recall:  74.19%; FB1:  73.63  936
              ORG: precision:  73.55%; recall:  67.19%; FB1:  70.23  1225
              PER: precision:  79.81%; recall:  72.53%; FB1:  76.00  1674
Epoch 16, Loss: 0.003172349494839595, time: 49.49557089805603s
validation error:
processed 152266 tokens with 5942 phrases; found: 5817 phrases; correct: 4554.
accuracy:  78.83%; (non-O)
accuracy:  98.49%; precision:  78.29%; recall:  76.64%; FB1:  77.46
              LOC: precision:  85.91%; recall:  83.61%; FB1:  84.74  1788
             MISC: precision:  74.62%; recall:  74.30%; FB1:  74.46  918
              ORG: precision:  71.13%; recall:  68.53%; FB1:  69.81  1292
              PER: precision:  77.74%; recall:  76.76%; FB1:  77.25  1819
Epoch 17, Loss: 0.002620459050971972, time: 47.30130100250244s
validation error:
processed 152266 tokens with 5942 phrases; found: 5804 phrases; correct: 4577.
accuracy:  79.24%; (non-O)
accuracy:  98.50%; precision:  78.86%; recall:  77.03%; FB1:  77.93
              LOC: precision:  87.66%; recall:  83.51%; FB1:  85.53  1750
             MISC: precision:  76.78%; recall:  73.86%; FB1:  75.29  887
              ORG: precision:  69.38%; recall:  70.47%; FB1:  69.92  1362
              PER: precision:  78.50%; recall:  76.93%; FB1:  77.71  1805
Epoch 18, Loss: 0.00238024852177742, time: 47.947713136672974s
validation error:
processed 152266 tokens with 5942 phrases; found: 5805 phrases; correct: 4541.
accuracy:  78.55%; (non-O)
accuracy:  98.47%; precision:  78.23%; recall:  76.42%; FB1:  77.31
              LOC: precision:  85.42%; recall:  84.21%; FB1:  84.81  1811
             MISC: precision:  76.91%; recall:  72.99%; FB1:  74.90  875
              ORG: precision:  69.67%; recall:  70.25%; FB1:  69.96  1352
              PER: precision:  78.04%; recall:  74.86%; FB1:  76.42  1767
Epoch 19, Loss: 0.002281301094626542, time: 48.1134819984436s
validation error:
processed 152266 tokens with 5942 phrases; found: 5658 phrases; correct: 4538.
accuracy:  78.69%; (non-O)
accuracy:  98.56%; precision:  80.21%; recall:  76.37%; FB1:  78.24
              LOC: precision:  87.74%; recall:  84.16%; FB1:  85.91  1762
             MISC: precision:  77.63%; recall:  73.75%; FB1:  75.64  876
              ORG: precision:  75.79%; recall:  67.93%; FB1:  71.65  1202
              PER: precision:  77.06%; recall:  76.06%; FB1:  76.56  1818
Epoch 20, Loss: 0.002566172640349991, time: 48.22484111785889s
validation error:
processed 152266 tokens with 5942 phrases; found: 5651 phrases; correct: 4545.
accuracy:  78.65%; (non-O)
accuracy:  98.55%; precision:  80.43%; recall:  76.49%; FB1:  78.41
              LOC: precision:  88.78%; recall:  83.56%; FB1:  86.09  1729
```

```
         MISC: precision:  78.97%; recall:  74.95%; FB1:  76.91  875
          ORG: precision:  74.96%; recall:  67.64%; FB1:  71.11  1210
          PER: precision:  76.86%; recall:  76.66%; FB1:  76.76  1837
```

[1162]:
```python
print(f"Validation: precision = {precision}, recall = {recall}, f1 = {f1}")
```

```
precision = 80.42824278888693, recall = 76.48939750925614, f1 =
78.40938497369102
```

[1151]:
```python
# SAVE THE MODEL
torch.save(model.state_dict(), 'task1.pth')
```

[444]:
```python
# Example reversed_ner_tags dictionary
reversed_ner_tags = {
    0: 'O',
    1: 'B-PER',
    2: 'I-PER',
    3: 'B-ORG',
    4: 'I-ORG',
    5: 'B-LOC',
    6: 'I-LOC',
    7: 'B-MISC',
    8: 'I-MISC'
}

# Example tensor with shape (32, 36)
tensor = torch.randint(0, 9, (32, 36))  # Random integers between 0 and 8

# Map tensor elements using reversed_ner_tags
mapped_tensor = [[reversed_ner_tags[item.item()] for item in row] for row in␣
 ↪tensor]
```

[398]:
```python
ner_tags = {'O': 0, 'B-PER': 1, 'I-PER': 2, 'B-ORG': 3, 'I-ORG': 4, 'B-LOC': 5,␣
 ↪'I-LOC': 6, 'B-MISC': 7, 'I-MISC': 8}

reversed_ner_tags = {value: key for key, value in ner_tags.items()}
reversed_ner_tags
```

[398]:
```
{0: 'O',
 1: 'B-PER',
 2: 'I-PER',
 3: 'B-ORG',
 4: 'I-ORG',
 5: 'B-LOC',
 6: 'I-LOC',
 7: 'B-MISC',
 8: 'I-MISC'}
```

[1149]:
```
#evaluation
def eval(model, loader):
    model.eval()
    all_preds, all_labels = [], []
    with torch.no_grad():
        for batch in loader:
            inputs, targets = batch
            outputs = model(inputs)
            _, preds = torch.max(outputs, -1)
            preds_converted = [[reversed_ner_tags[item.item()] for item in row]␣
 ↪for row in preds]
            targets_converted = [[reversed_ner_tags[item.item()] for item in␣
 ↪row] for row in targets]
            all_preds.extend(preds_converted)
            all_labels.extend(targets_converted)
    # all_preds = list(chain.from_iterable(all_preds))
    # all_labels = list(chain.from_iterable(all_labels))
    # all_labels = torch.cat(all_labels)
    all_preds = itertools.chain(*all_preds)
    all_labels =itertools.chain(*all_labels)
    result = evaluate(all_labels, all_preds,verbose=True)
    precision, recall, f1 = result[0], result[1],result[2]
    return precision, recall, f1
```

[1163]:
```
print('Test: ')
precision, recall, f1 = eval(model, test_loader)
```

```
Test:
processed 146937 tokens with 5648 phrases; found: 5146 phrases; correct: 3710.
accuracy:  70.02%; (non-O)
accuracy:  97.95%; precision:  72.09%; recall:  65.69%; FB1:  68.74
              LOC: precision:  84.52%; recall:  75.30%; FB1:  79.64  1486
             MISC: precision:  64.47%; recall:  62.82%; FB1:  63.64  684
              ORG: precision:  67.13%; recall:  57.80%; FB1:  62.12  1430
              PER: precision:  68.11%; recall:  65.12%; FB1:  66.58  1546
```

[1164]:
```
print(f"Test: precision = {precision}, recall = {recall}, f1 = {f1}")
```

```
Test: precision = 72.09483093664983, recall = 65.68696883852692, f1 =
68.74189364461739
```

### 0.0.5 Solution for the task 1

1. Hyperparameters:

- vocab_size = 8128
- embedding_dim = 100
- hidden_dim = 256
- output_dim = 128

- num_layers = 1
- dropout = 0.33
- optimizer learning rate= 0.001
- batch_size = 64

2. Solution: At first, I created a vocab that maps all the tokens from the training set to a number, and I gave up the tokens that appeared less than 3 times. Secondly, I custimized a dataset class so that each batch will conatin (input_ids, ner_tags). Next, I used padding_sequence to customize the padding value of 0 in input and 9 in ner_tags. Why do I pad here? I need to make sure for each batch, which contains 32 samples, will have the max_length within one batch. Thirdly, I designed my bilstm model. The model will firstly embed all the inputs to 100-dim vectors and then throw the vectors to the lstm layer. Through elu, dropout, and one more linear layer, it model will predict the name entity for each token in samples.
3. Questions and answers:

- What are the precision, recall, and F1 score on the validation data?
- precision = 80.42824278888693, recall = 76.48939750925614, f1 = 78.40938497369102- What are the precision, recall, and F1 score on the test data?
- precision = 72.09483093664983, recall = 65.68696883852692, f1 = 68.74189364461739

## 0.1 Task 2: Glove Embedding

### 0.1.1 Load Glove Embedding

[31]:
```python
# Define a function to load GloVe embeddings from a file
def load_glove_embeddings(file_path):
    embeddings_index = {}
    with open(file_path, encoding="utf-8") as f:
        for line in f:
            values = line.split()
            word = values[0]
            coefs = np.asarray(values[1:], dtype="float32")
            embeddings_index[word] = coefs
    return embeddings_index

# Specify the path to your downloaded "glove.6B.100d.txt" file
glove_file_path = "glove.6B.100d"

# Load GloVe embeddings into memory
glove_embeddings = load_glove_embeddings(glove_file_path)
```

### 0.1.2 Create Glove Idx

[1027]:
```python
def convert_word_to_glove_ids(sample):
    tokens = sample['tokens']
    glove_ids =[]
    for token in tokens:
        token = token.lower()
        indices = np.where(vocab_npa == token)
```

```
        if indices[0].size > 0:
            index = indices[0][0]
        else:
            index = 1
        glove_ids.append(index)
    sample['glove_ids'] = glove_ids
    return sample
dataset = dataset.map(convert_word_to_glove_ids)
```

Map: 0%| | 0/14041 [00:00<?, ? examples/s]

Map: 0%| | 0/3250 [00:00<?, ? examples/s]

Map: 0%| | 0/3453 [00:00<?, ? examples/s]

### 0.1.3 Customize the layer

[831]:
```
#convert glove into a layer
vocab,embeddings = [],[]
with open('glove.6B.100d',encoding="utf-8") as fi:
    full_content = fi.read().strip().split('\n')
for i in range(len(full_content)):
    i_word = full_content[i].split(' ')[0]
    i_embeddings = [float(val) for val in full_content[i].split(' ')[1:]]
    vocab.append(i_word)
    embeddings.append(i_embeddings)
```

[832]:
```
vocab_npa = np.array(vocab)
embs_npa = np.array(embeddings)

#insert '<pad>' and '<unk>' tokens at start of vocab_npa.
vocab_npa = np.insert(vocab_npa, 0, '<pad>')
vocab_npa = np.insert(vocab_npa, 1, '<unk>')
print(vocab_npa[:10])

pad_emb_npa = np.zeros((1,embs_npa.shape[1]))   #embedding for '<pad>' token.
unk_emb_npa = np.mean(embs_npa,axis=0,keepdims=True)    #embedding for '<unk>'␣
↪token.

#insert embeddings for pad and unk tokens at top of embs_npa.
embs_npa = np.vstack((pad_emb_npa,unk_emb_npa,embs_npa))
print(embs_npa.shape)
```

```
['<pad>' '<unk>' 'the' ',' '.' 'of' 'to' 'and' 'in' 'a']
(400002, 100)
```

[877]:
```
import torch
my_embedding_layer = torch.nn.Embedding.from_pretrained(torch.
↪from_numpy(embs_npa).float(),freeze=True)
```

```
assert my_embedding_layer.weight.shape == embs_npa.shape
print(my_embedding_layer.weight.shape)
```

```
torch.Size([400002, 100])
```

### 0.1.4 Make Glove case-sensitive – creating another feature

[858]:
```
#add features to the dataloader
#case 0: lower case - no uppercase
#case 1: first word is uppercase
#case 2: whole word is uppeercase
#case 3: others: e.g. ","
def capital_case(word):
    if word.islower():
        return 0
    elif word.isupper():
        return 2
    elif word.istitle():
        return 1
    else: return 3

def convert_word_to_capital_case(sample):
    capitals = [capital_case(word) for word in sample['tokens'] ]
    sample['capitals'] =capitals
    return sample

dataset = dataset.map(convert_word_to_capital_case)
```

```
Map:   0%|          | 0/14041 [00:00<?, ? examples/s]

Map:   0%|          | 0/3250 [00:00<?, ? examples/s]

Map:   0%|          | 0/3453 [00:00<?, ? examples/s]
```

[1023]:
```
dataset['train'][2]
```

[1023]:
```
{'id': '2',
 'tokens': ['BRUSSELS', '1996-08-22'],
 'pos_tags': [22, 11],
 'chunk_tags': [11, 12],
 'ner_tags': [5, 0],
 'input_ids': [12, 13],
 'capitals': [2, 3],
 'glove_ids': [1, 1]}
```

### 0.1.5 Padding – glove embedding

[1165]:
```
import pandas as pd
import torch
from torch.utils.data import Dataset

# Create a custom Dataset class
class CustomDataset(Dataset):

    def __init__(self,data):
        self.data = data

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

    def __getitem__(self, index):
        label = torch.tensor(self.data[index]['ner_tags'], dtype=torch.long )
        glove_ids = torch.tensor(self.data[index]['glove_ids'], dtype=torch.
 ↪long)
        capital = torch.tensor(self.data[index]['capitals'], dtype=torch.long)

        return label, glove_ids, capital

# Create an instance of the CustomDataset
dataset_train = CustomDataset(dataset['train'])
dataset_test = CustomDataset(dataset['test'])
dataset_val = CustomDataset(dataset['validation'])

# Example: Accessing a single sample
print(dataset_train[0])
```

```
(tensor([3, 0, 7, 0, 0, 0, 7, 0, 0]), tensor([  646,  7580,   516,   582,     6,
5262,   299, 10240,     4]), tensor([2, 0, 1, 0, 0, 0, 1, 0, 3]))
```

[1166]:
```
def custom_collate(batch):
    label, glove_ids, capital = zip(*batch)
    padded_label = pad_sequence(label, batch_first=True, padding_value=9 )
    padded_glove_ids = pad_sequence(glove_ids, batch_first=True,␣
 ↪padding_value=0 )
    padded_capital = pad_sequence(capital, batch_first=True, padding_value=4 )
    return padded_glove_ids, padded_capital, padded_label
```

[1167]:
```
batch_size = 64
train_loader = DataLoader(dataset_train, batch_size=batch_size, collate_fn=␣
 ↪custom_collate, shuffle=True)
test_loader = DataLoader(dataset_test, batch_size=batch_size, collate_fn=␣
 ↪custom_collate, shuffle=False)
```

```
val_loader = DataLoader(dataset_val, batch_size=batch_size, collate_fn=
    ↪custom_collate, shuffle=False)
```

[1174]:

```
class BiLSTMNER(nn.Module):
    def __init__(self,hidden_dim, output_dim, num_layers, dropout):
        super(BiLSTMNER, self).__init__()
        self.embedding = my_embedding_layer
        self.capital_layer = nn.
    ↪Embedding(num_embeddings=5,embedding_dim=20,padding_idx=4)
        self.bilstm = nn.LSTM(input_size=120, hidden_size=hidden_dim,
    ↪num_layers=num_layers,
                              batch_first=True, bidirectional=True)
        self.dropout = nn.Dropout(dropout)
        self.linear = nn.Linear(hidden_dim * 2, output_dim,dtype=torch.float32)
        self.elu = nn.ELU()
        self.classifier = nn.Linear(output_dim, num_tags,dtype=torch.float32)
    ↪# num_tags is the number of unique NER tags

    def forward(self, x, capital):
        x = self.embedding(x.int())
        capital = self.capital_layer(capital.int())
        x = torch.cat([x, capital], dim=2)
        x, _ = self.bilstm(x)
        x = self.dropout(x)
        x = self.linear(x)
        x = self.elu(x)
        x = self.classifier(x)
        return x


#initialize
num_tags = 9
model = BiLSTMNER(256,128, 1, 0.33)
optimizer = optim.Adam(model.parameters(), lr=0.001)
loss_function = nn.CrossEntropyLoss(ignore_index=9)


#training
num_epochs = 20
print('start training')
for epoch in range(num_epochs):
    start_time = time.time()
    model.train()
    total_loss = 0
    for batch in train_loader:
        optimizer.zero_grad()
        inputs, capitals ,targets = batch
```

```python
        outputs = model(inputs, capitals)
        batch_size = inputs.size()[-1]
        #From the instruction of CrossEntropy, we need to change the format of␣
↪outputs
        loss = loss_function(outputs.permute(0,2,1), targets)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    end_time = time.time()
    print(f'Epoch {epoch + 1}, Loss: {total_loss / len(train_loader)}, time:␣
↪{end_time-start_time}s')
    print('validation error: ')
    precision, recall, f1 = eval(model, val_loader)
```

```
start training
Epoch 1, Loss: 0.2787948575378819, time: 59.194642066955566s
validation error:
processed 51362 tokens with 5942 phrases; found: 6183 phrases; correct: 5010.
accuracy:  85.81%; (non-O)
accuracy:  97.16%; precision:  81.03%; recall:  84.32%; FB1:  82.64
              LOC: precision:  83.66%; recall:  90.85%; FB1:  87.11  1995
             MISC: precision:  68.96%; recall:  74.95%; FB1:  71.83  1002
              ORG: precision:  72.70%; recall:  70.69%; FB1:  71.68  1304
              PER: precision:  90.44%; recall:  92.40%; FB1:  91.41  1882
Epoch 2, Loss: 0.0857482789422978, time: 59.39733099937439s
validation error:
processed 51362 tokens with 5942 phrases; found: 6033 phrases; correct: 5285.
accuracy:  89.63%; (non-O)
accuracy:  97.99%; precision:  87.60%; recall:  88.94%; FB1:  88.27
              LOC: precision:  92.77%; recall:  92.16%; FB1:  92.46  1825
             MISC: precision:  79.46%; recall:  79.28%; FB1:  79.37  920
              ORG: precision:  78.05%; recall:  84.04%; FB1:  80.93  1444
              PER: precision:  94.03%; recall:  94.14%; FB1:  94.09  1844
Epoch 3, Loss: 0.06427998816255819, time: 58.05035185813904s
validation error:
processed 51362 tokens with 5942 phrases; found: 6028 phrases; correct: 5366.
accuracy:  91.06%; (non-O)
accuracy:  98.25%; precision:  89.02%; recall:  90.31%; FB1:  89.66
              LOC: precision:  94.65%; recall:  92.49%; FB1:  93.56  1795
             MISC: precision:  80.36%; recall:  82.54%; FB1:  81.43  947
              ORG: precision:  81.43%; recall:  85.98%; FB1:  83.64  1416
              PER: precision:  93.74%; recall:  95.17%; FB1:  94.45  1870
Epoch 4, Loss: 0.05143080727959221, time: 58.354299783706665s
validation error:
processed 51362 tokens with 5942 phrases; found: 6084 phrases; correct: 5425.
accuracy:  92.00%; (non-O)
accuracy:  98.33%; precision:  89.17%; recall:  91.30%; FB1:  90.22
```

```
             LOC: precision:  93.02%; recall:  95.75%; FB1:  94.37  1891
            MISC: precision:  78.62%; recall:  83.73%; FB1:  81.09  982
             ORG: precision:  84.84%; recall:  83.89%; FB1:  84.36  1326
             PER: precision:  93.85%; recall:  96.04%; FB1:  94.93  1885
Epoch 5, Loss: 0.04187326981178061, time: 59.04758620262146s
validation error:
processed 51362 tokens with 5942 phrases; found: 6045 phrases; correct: 5450.
accuracy:  92.28%; (non-O)
accuracy:  98.47%; precision:  90.16%; recall:  91.72%; FB1:  90.93
             LOC: precision:  93.03%; recall:  95.16%; FB1:  94.08  1879
            MISC: precision:  83.41%; recall:  82.86%; FB1:  83.13  916
             ORG: precision:  84.62%; recall:  87.40%; FB1:  85.99  1385
             PER: precision:  94.69%; recall:  95.87%; FB1:  95.28  1865
Epoch 6, Loss: 0.03367272468114441, time: 58.262818813323975s
validation error:
processed 51362 tokens with 5942 phrases; found: 6143 phrases; correct: 5483.
accuracy:  93.15%; (non-O)
accuracy:  98.44%; precision:  89.26%; recall:  92.28%; FB1:  90.74
             LOC: precision:  94.36%; recall:  95.59%; FB1:  94.97  1861
            MISC: precision:  79.81%; recall:  82.75%; FB1:  81.26  956
             ORG: precision:  82.64%; recall:  89.49%; FB1:  85.93  1452
             PER: precision:  94.13%; recall:  95.77%; FB1:  94.94  1874
Epoch 7, Loss: 0.027345544093457814, time: 59.46216917037964s
validation error:
processed 51362 tokens with 5942 phrases; found: 6015 phrases; correct: 5484.
accuracy:  92.49%; (non-O)
accuracy:  98.54%; precision:  91.17%; recall:  92.29%; FB1:  91.73
             LOC: precision:  94.70%; recall:  95.32%; FB1:  95.01  1849
            MISC: precision:  82.86%; recall:  86.01%; FB1:  84.41  957
             ORG: precision:  87.71%; recall:  87.84%; FB1:  87.78  1343
             PER: precision:  94.43%; recall:  95.66%; FB1:  95.04  1866
Epoch 8, Loss: 0.021476747551721267, time: 58.77496004104614s
validation error:
processed 51362 tokens with 5942 phrases; found: 6044 phrases; correct: 5502.
accuracy:  92.69%; (non-O)
accuracy:  98.55%; precision:  91.03%; recall:  92.60%; FB1:  91.81
             LOC: precision:  94.57%; recall:  95.75%; FB1:  95.16  1860
            MISC: precision:  85.90%; recall:  83.95%; FB1:  84.91  901
             ORG: precision:  84.22%; recall:  89.93%; FB1:  86.98  1432
             PER: precision:  95.25%; recall:  95.71%; FB1:  95.48  1851
Epoch 9, Loss: 0.01826574724400416, time: 60.954275131225586s
validation error:
processed 51362 tokens with 5942 phrases; found: 6081 phrases; correct: 5504.
accuracy:  93.13%; (non-O)
accuracy:  98.58%; precision:  90.51%; recall:  92.63%; FB1:  91.56
             LOC: precision:  95.46%; recall:  94.99%; FB1:  95.23  1828
            MISC: precision:  82.56%; recall:  84.71%; FB1:  83.62  946
             ORG: precision:  84.06%; recall:  90.83%; FB1:  87.31  1449
```

```
             PER: precision:  94.73%; recall:  95.55%; FB1:  95.14  1858
Epoch 10, Loss: 0.013936886461239986, time: 64.49255204200745s
validation error:
processed 51362 tokens with 5942 phrases; found: 6052 phrases; correct: 5498.
accuracy:  92.89%; (non-O)
accuracy:  98.58%; precision:  90.85%; recall:  92.53%; FB1:  91.68
             LOC: precision:  95.55%; recall:  94.67%; FB1:  95.11  1820
            MISC: precision:  81.27%; recall:  87.53%; FB1:  84.28  993
             ORG: precision:  86.99%; recall:  87.77%; FB1:  87.38  1353
             PER: precision:  94.11%; recall:  96.36%; FB1:  95.23  1886
Epoch 11, Loss: 0.010753243909725412, time: 63.68161940574646s
validation error:
processed 51362 tokens with 5942 phrases; found: 6062 phrases; correct: 5484.
accuracy:  92.71%; (non-O)
accuracy:  98.53%; precision:  90.47%; recall:  92.29%; FB1:  91.37
             LOC: precision:  94.12%; recall:  95.92%; FB1:  95.01  1872
            MISC: precision:  83.39%; recall:  86.01%; FB1:  84.68  951
             ORG: precision:  84.46%; recall:  87.55%; FB1:  85.98  1390
             PER: precision:  94.92%; recall:  95.28%; FB1:  95.10  1849
Epoch 12, Loss: 0.009369476515249433, time: 61.54601192474365s
validation error:
processed 51362 tokens with 5942 phrases; found: 6029 phrases; correct: 5496.
accuracy:  92.99%; (non-O)
accuracy:  98.62%; precision:  91.16%; recall:  92.49%; FB1:  91.82
             LOC: precision:  95.07%; recall:  95.43%; FB1:  95.25  1844
            MISC: precision:  81.38%; recall:  86.77%; FB1:  83.99  983
             ORG: precision:  87.74%; recall:  87.02%; FB1:  87.38  1330
             PER: precision:  94.87%; recall:  96.42%; FB1:  95.64  1872
Epoch 13, Loss: 0.007336435311431573, time: 62.67510199546814s
validation error:
processed 51362 tokens with 5942 phrases; found: 6031 phrases; correct: 5527.
accuracy:  93.33%; (non-O)
accuracy:  98.66%; precision:  91.64%; recall:  93.02%; FB1:  92.32
             LOC: precision:  94.47%; recall:  96.79%; FB1:  95.62  1882
            MISC: precision:  83.98%; recall:  84.71%; FB1:  84.34  930
             ORG: precision:  87.96%; recall:  89.86%; FB1:  88.90  1370
             PER: precision:  95.35%; recall:  95.71%; FB1:  95.53  1849
Epoch 14, Loss: 0.006210239743284712, time: 63.491557121276855s
validation error:
processed 51362 tokens with 5942 phrases; found: 6091 phrases; correct: 5548.
accuracy:  93.79%; (non-O)
accuracy:  98.69%; precision:  91.09%; recall:  93.37%; FB1:  92.21
             LOC: precision:  95.59%; recall:  95.65%; FB1:  95.62  1838
            MISC: precision:  85.22%; recall:  86.33%; FB1:  85.78  934
             ORG: precision:  85.04%; recall:  91.13%; FB1:  87.98  1437
             PER: precision:  94.21%; recall:  96.25%; FB1:  95.22  1882
Epoch 15, Loss: 0.004891020257359328б, time: 89.17146420478821s
validation error:
```

```
processed 51362 tokens with 5942 phrases; found: 6044 phrases; correct: 5511.
accuracy:  93.11%; (non-O)
accuracy:  98.61%; precision:  91.18%; recall:  92.75%; FB1:  91.96
              LOC: precision:  94.52%; recall:  95.86%; FB1:  95.19  1863
             MISC: precision:  86.20%; recall:  86.01%; FB1:  86.10  920
              ORG: precision:  87.30%; recall:  87.62%; FB1:  87.46  1346
              PER: precision:  93.05%; recall:  96.74%; FB1:  94.86  1915
Epoch 16, Loss: 0.00643773535636931, time: 87.57236385345459s
validation error:
processed 51362 tokens with 5942 phrases; found: 6053 phrases; correct: 5531.
accuracy:  93.32%; (non-O)
accuracy:  98.64%; precision:  91.38%; recall:  93.08%; FB1:  92.22
              LOC: precision:  94.51%; recall:  96.46%; FB1:  95.47  1875
             MISC: precision:  84.08%; recall:  85.36%; FB1:  84.71  936
              ORG: precision:  88.36%; recall:  88.29%; FB1:  88.33  1340
              PER: precision:  94.01%; recall:  97.07%; FB1:  95.51  1902
Epoch 17, Loss: 0.004882407614059048, time: 63.116442918777466s
validation error:
processed 51362 tokens with 5942 phrases; found: 6020 phrases; correct: 5503.
accuracy:  92.97%; (non-O)
accuracy:  98.60%; precision:  91.41%; recall:  92.61%; FB1:  92.01
              LOC: precision:  94.92%; recall:  95.70%; FB1:  95.31  1852
             MISC: precision:  85.82%; recall:  84.71%; FB1:  85.26  910
              ORG: precision:  86.98%; recall:  89.19%; FB1:  88.07  1375
              PER: precision:  93.89%; recall:  95.98%; FB1:  94.93  1883
Epoch 18, Loss: 0.003682805795291312, time: 133.74659514427185s
validation error:
processed 51362 tokens with 5942 phrases; found: 6058 phrases; correct: 5526.
accuracy:  93.25%; (non-O)
accuracy:  98.63%; precision:  91.22%; recall:  93.00%; FB1:  92.10
              LOC: precision:  94.57%; recall:  96.62%; FB1:  95.58  1877
             MISC: precision:  84.57%; recall:  86.23%; FB1:  85.39  940
              ORG: precision:  87.42%; recall:  88.07%; FB1:  87.74  1351
              PER: precision:  93.92%; recall:  96.36%; FB1:  95.12  1890
Epoch 19, Loss: 0.003912460297711236, time: 155.75212383270264s
validation error:
processed 51362 tokens with 5942 phrases; found: 6046 phrases; correct: 5518.
accuracy:  93.14%; (non-O)
accuracy:  98.63%; precision:  91.27%; recall:  92.86%; FB1:  92.06
              LOC: precision:  94.93%; recall:  95.81%; FB1:  95.37  1854
             MISC: precision:  83.76%; recall:  86.12%; FB1:  84.92  948
              ORG: precision:  88.31%; recall:  87.92%; FB1:  88.12  1335
              PER: precision:  93.50%; recall:  96.91%; FB1:  95.17  1909
Epoch 20, Loss: 0.0027471507286959836, time: 69.84025812149048s
validation error:
processed 51362 tokens with 5942 phrases; found: 6044 phrases; correct: 5541.
accuracy:  93.47%; (non-O)
accuracy:  98.67%; precision:  91.68%; recall:  93.25%; FB1:  92.46
```

```
             LOC: precision:  94.80%; recall:  96.30%; FB1:  95.54  1866
            MISC: precision:  85.01%; recall:  86.12%; FB1:  85.56  934
             ORG: precision:  88.16%; recall:  89.41%; FB1:  88.78  1360
             PER: precision:  94.43%; recall:  96.58%; FB1:  95.49  1884
```

[1179]:
```python
# SAVE THE MODEL
torch.save(model.state_dict(), 'task2.pth')
```

[1176]:
```python
print(f"Validation: precision = {precision}, recall = {recall}, f1 = {f1}")
```

```
Validation: precision = 91.67769688947716, recall = 93.25143049478291, f1 =
92.45786751209747
```

[1172]:
```python
ner_tags = {'O': 0, 'B-PER': 1, 'I-PER': 2, 'B-ORG': 3, 'I-ORG': 4, 'B-LOC': 5,␣
↪'I-LOC': 6, 'B-MISC': 7, 'I-MISC': 8, '<PAD>':9}

reversed_ner_tags = {value: key for key, value in ner_tags.items()}
reversed_ner_tags
```

[1172]:
```
{0: 'O',
 1: 'B-PER',
 2: 'I-PER',
 3: 'B-ORG',
 4: 'I-ORG',
 5: 'B-LOC',
 6: 'I-LOC',
 7: 'B-MISC',
 8: 'I-MISC',
 9: '<PAD>'}
```

[1177]:
```python
#evaluation
def eval(model, loader):
    model.eval()
    all_preds, all_labels = [], []
    with torch.no_grad():
        for batch in loader:
            inputs, capitals ,targets = batch
            #get rid of paddings on targets
            label_unpad = targets
            mask = label_unpad != 9
            label_unpad = label_unpad[mask]

            outputs = model(inputs,capitals)
            _, preds = torch.max(outputs, -1)
            #get rid of paddings on pred
            preds = preds[mask]

            preds_converted = [reversed_ner_tags[elem.item()] for elem in preds]
```

```
            targets_converted = [reversed_ner_tags[elem.item()] for elem in␣
↪label_unpad]
            all_preds.extend(preds_converted)
            all_labels.extend(targets_converted)
    # all_preds = list(chain.from_iterable(all_preds))
    # all_labels = list(chain.from_iterable(all_labels))
    # all_labels = torch.cat(all_labels)
    # all_preds = itertools.chain(*all_preds)
    # all_labels =itertools.chain(*all_labels)
    result = evaluate(all_labels, all_preds,verbose=True)
    precision, recall, f1 = result[0], result[1],result[2]
    return precision, recall, f1

# print('Test: ')
# precision, recall, f1 = eval(model, test_loader)
```

[1178]:
```
print(f"Test: precision = {precision}, recall = {recall}, f1 = {f1}")
```

```
Test: precision = 91.67769688947716, recall = 93.25143049478291, f1 =
92.45786751209747
```

#### 0.1.6 Solution for task2

1. Hyperparameters:

- embedding_dim = 100
- hidden_dim = 256
- output_dim = 128
- num_layers = 1
- dropout = 0.33
- optimizer learning rate= 0.001
- ignore_index = 9
- batch_size = 64

2. Solution: At first, I loaded the glove embedding and convert it into two arrays. One records all the indices and the other one records the 100-d embeddings for all the tokens. Secondly, since the glove is not case-sensitive, I tried to divide tokens into 4 cases (0: lowercase 1: some uppercases 2: all uppercases 3: lowercase and uppercase are the same). So, I added a new list to the dataset. Thirdly, I mapped all the tokens into indices in the glove embedding. So, I added one more list to the dataset. Forthly, I created a new customized dataset that each batch contains (glove_ids, capitalize, ner_tag). And similiar to the task, I padded all of them while creating the dataloaders. To be notified, I padded 9 to the ner_tag since it is a number that has not been used. I padded the capitalize with 4, which is not used either. Fifthly, I threw the batches into the model, which has the similar structure to the task 1. However, I added one more embedding layer such that the feature capitalize will be converted into 20-d vector and be added to the original 100-d layer. So, the input will become a 120-d vector. Through elu, dropout, and one more linear layer, it model will predict the name entity for each token in samples.
3. Questions and answers:

- What is the precision, recall, and F1 score on the validation data?
- precision = 91.67769688947716, recall = 93.25143049478291, f1 = 92.45786751209747- What are the precision, recall, and F1 score on the test data?
- precision = 91.67769688947716, recall = 93.25143049478291, f1 = 92.45786751209747- BiLSTM with Glove Embeddings outperforms the model without. Can you provide a rationale for this?
- At first, the glove is a bigger vocab than the word2idx, so it will map less unknown words. Secondly, since I added a new embedding layer, the model can better capture whether the word has been capitalized.

## 0.2 Task3: Transformer

[1180]:
```
import pandas as pd
import torch
from torch.utils.data import Dataset

class CustomDataset(Dataset):

    def __init__(self,data):
        self.data = data

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

    def __getitem__(self, index):
        input = torch.tensor(self.data[index]['input_ids'], dtype=torch.long )
        target = torch.tensor(self.data[index]['ner_tags'], dtype=torch.long)

        return input, target

# Create an instance of the CustomDataset
dataset_train = CustomDataset(dataset['train'])
dataset_test = CustomDataset(dataset['test'])
dataset_val = CustomDataset(dataset['validation'])

# Example: Accessing a single sample
print(dataset_train[2])
```

```
(tensor([12, 13]), tensor([5, 0]))
```

[1181]:
```
def custom_collate(batch):
    input_ids, label = zip(*batch)
    padded_label = pad_sequence(label, batch_first=True, padding_value=9 )
    padded_input_ids = pad_sequence(input_ids, batch_first=True,␣
↪padding_value=0 )
    return padded_input_ids, padded_label
```

[1182]:
```
batch_size = 32
train_loader = DataLoader(dataset_train, batch_size=batch_size, collate_fn=␣
↪custom_collate, shuffle=True)
test_loader = DataLoader(dataset_test, batch_size=batch_size, collate_fn=␣
↪custom_collate, shuffle=False)
val_loader = DataLoader(dataset_val, batch_size=batch_size, collate_fn=␣
↪custom_collate, shuffle=False)
```

[1183]:
```
# inspect the trainloader
for batch in train_loader:
    inputs, labels = batch
    break
```

[1184]:
```
import torch
import torch.nn as nn

class TransformerNERModel(nn.Module):
    def __init__(self, vocab_size, tag_vocab_size, embed_size=128, num_heads=8,␣
↪max_seq_length=128, ff_dim=128, num_encoder_layers=6,
                 dropout=0.33):
        super(TransformerNERModel, self).__init__()

        # Token embedding layer
        self.embedding = TokenEmbedding(vocab_size, embed_size)

        # Positional encoding
        self.positional_encoder = PositionalEncoding(emb_size= embed_size,␣
↪maxlen=max_seq_length)

        # Transformer Encoder
        self.transformer_encoder = nn.TransformerEncoder(
            nn.TransformerEncoderLayer(
                d_model=embed_size,
                nhead=num_heads,
                dim_feedforward=ff_dim,
                batch_first=True
            ),
            num_layers=num_encoder_layers,
        )

        # Linear layer for classification
        self.fc = nn.Linear(embed_size, tag_vocab_size)

    def forward(self, src, src_padding_mask):
        # Token embedding
        x = self.embedding(src)
```

```
        # Add positional encoding
        x = self.positional_encoder(x)

        # Transformer encoder
        x = self.transformer_encoder(x, src_key_padding_mask=src_padding_mask)

        # Final linear layer for classification
        x = self.fc(x)

        return x

class PositionalEncoding(nn.Module):
    def __init__(self,
                 emb_size: int,
                 dropout: float =0.33,
                 maxlen: int = 5000):
        super(PositionalEncoding, self).__init__()
        den = torch.exp(- torch.arange(0, emb_size, 2)* math.log(10000) /␣
 ↪emb_size)
        pos = torch.arange(0, maxlen).reshape(maxlen, 1)
        pos_embedding = torch.zeros((maxlen, emb_size))
        pos_embedding[:, 0::2] = torch.sin(pos * den)
        pos_embedding[:, 1::2] = torch.cos(pos * den)
        pos_embedding = pos_embedding.unsqueeze(-2)

        self.dropout = nn.Dropout(dropout)
        self.register_buffer('pos_embedding', pos_embedding)

    def forward(self, token_embedding):
        return self.dropout(token_embedding + self.pos_embedding[:
 ↪token_embedding.size(0), :])

class TokenEmbedding(nn.Module):
    def __init__(self, vocab_size: int, emb_size):
        super(TokenEmbedding, self).__init__()
        self.embedding = nn.Embedding(vocab_size, emb_size)
        self.emb_size = emb_size

    def forward(self, tokens):
        return self.embedding(tokens.long()) * math.sqrt(self.emb_size)

# Initialize the model
vocab_size = max(word2idx.values())+1# Your vocabulary size
tag_vocab_size = 9 # Your tag vocabulary size
model = TransformerNERModel(vocab_size, tag_vocab_size)

criterion = nn.CrossEntropyLoss(ignore_index=9)
```

```
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

[1187]:
```
# Training loop
num_epochs = 25
for epoch in range(num_epochs):
    model.train()
    total_loss = 0.0

    # Iterate over your training data in batches
    for batch_idx, (inputs, targets) in enumerate(train_loader):
        # Zero the gradients
        optimizer.zero_grad()

        # Forward pass + src_padding_mask
        src_padding_mask = (inputs == 0).float()
        outputs = model(inputs, src_padding_mask= src_padding_mask)

        # Flatten the outputs and targets for the loss calculation
        outputs = outputs.view(-1, 9)
        targets = targets.view(-1)

        # Calculate the loss
        loss = criterion(outputs, targets)

        # Backpropagation
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    # Print the average loss for this epoch
    avg_loss = total_loss / len(train_loader)
    print(f"Epoch [{epoch + 1}/{num_epochs}] - Loss: {avg_loss:.4f}")

    print('validation error: ')
    precision, recall, f1 = eval(model, val_loader)
```

```
Epoch [1/20] - Loss: 0.5256
validation error:
processed 51362 tokens with 5942 phrases; found: 3827 phrases; correct: 1857.
accuracy:  29.40%; (non-O)
accuracy:  87.05%; precision:  48.52%; recall:  31.25%; FB1:  38.02
              LOC: precision:  58.47%; recall:  52.04%; FB1:  55.07  1635
             MISC: precision:  64.33%; recall:  23.86%; FB1:  34.81  342
              ORG: precision:  53.95%; recall:  17.30%; FB1:  26.20  430
              PER: precision:  31.62%; recall:  24.38%; FB1:  27.53  1420
Epoch [2/20] - Loss: 0.4477
validation error:
```

```
processed 51362 tokens with 5942 phrases; found: 3723 phrases; correct: 2158.
accuracy:  34.28%; (non-O)
accuracy:  88.53%; precision:  57.96%; recall:  36.32%; FB1:  44.66
              LOC: precision:  75.40%; recall:  54.38%; FB1:  63.19  1325
             MISC: precision:  72.88%; recall:  41.97%; FB1:  53.27  531
              ORG: precision:  48.84%; recall:  29.83%; FB1:  37.04  819
              PER: precision:  35.50%; recall:  20.20%; FB1:  25.74  1048
Epoch [3/20] - Loss: 0.3974
validation error:
processed 51362 tokens with 5942 phrases; found: 4277 phrases; correct: 2363.
accuracy:  39.20%; (non-O)
accuracy:  89.35%; precision:  55.25%; recall:  39.77%; FB1:  46.25
              LOC: precision:  80.41%; recall:  57.43%; FB1:  67.01  1312
             MISC: precision:  67.03%; recall:  53.15%; FB1:  59.29  731
              ORG: precision:  51.54%; recall:  31.25%; FB1:  38.90  813
              PER: precision:  28.08%; recall:  21.66%; FB1:  24.46  1421
Epoch [4/20] - Loss: 0.3614
validation error:
processed 51362 tokens with 5942 phrases; found: 5168 phrases; correct: 2721.
accuracy:  44.73%; (non-O)
accuracy:  89.73%; precision:  52.65%; recall:  45.79%; FB1:  48.98
              LOC: precision:  78.47%; recall:  61.89%; FB1:  69.20  1449
             MISC: precision:  67.09%; recall:  57.92%; FB1:  62.17  796
              ORG: precision:  55.21%; recall:  34.75%; FB1:  42.65  844
              PER: precision:  28.09%; recall:  31.70%; FB1:  29.79  2079
Epoch [5/20] - Loss: 0.3321
validation error:
processed 51362 tokens with 5942 phrases; found: 5498 phrases; correct: 2887.
accuracy:  49.01%; (non-O)
accuracy:  90.45%; precision:  52.51%; recall:  48.59%; FB1:  50.47
              LOC: precision:  82.54%; recall:  61.51%; FB1:  70.49  1369
             MISC: precision:  69.62%; recall:  57.92%; FB1:  63.23  767
              ORG: precision:  45.41%; recall:  47.58%; FB1:  46.47  1405
              PER: precision:  29.89%; recall:  31.76%; FB1:  30.80  1957
Epoch [6/20] - Loss: 0.3095
validation error:
processed 51362 tokens with 5942 phrases; found: 5344 phrases; correct: 2915.
accuracy:  49.69%; (non-O)
accuracy:  90.79%; precision:  54.55%; recall:  49.06%; FB1:  51.66
              LOC: precision:  76.19%; recall:  65.16%; FB1:  70.25  1571
             MISC: precision:  66.55%; recall:  61.50%; FB1:  63.92  852
              ORG: precision:  50.60%; recall:  37.96%; FB1:  43.37  1006
              PER: precision:  33.52%; recall:  34.85%; FB1:  34.18  1915
Epoch [7/20] - Loss: 0.2901
validation error:
processed 51362 tokens with 5942 phrases; found: 5740 phrases; correct: 2861.
accuracy:  49.97%; (non-O)
accuracy:  90.56%; precision:  49.84%; recall:  48.15%; FB1:  48.98
```

```
             LOC: precision:  77.28%; recall:  66.09%; FB1:  71.24  1571
            MISC: precision:  64.77%; recall:  61.61%; FB1:  63.15  877
             ORG: precision:  50.54%; recall:  31.17%; FB1:  38.56  827
             PER: precision:  26.82%; recall:  35.88%; FB1:  30.69  2465
Epoch [8/20] - Loss: 0.2772
validation error:
processed 51362 tokens with 5942 phrases; found: 5131 phrases; correct: 3051.
accuracy:  51.10%; (non-O)
accuracy:  91.32%; precision:  59.46%; recall:  51.35%; FB1:  55.11
             LOC: precision:  82.48%; recall:  66.14%; FB1:  73.41  1473
            MISC: precision:  68.18%; recall:  63.45%; FB1:  65.73  858
             ORG: precision:  54.11%; recall:  48.10%; FB1:  50.93  1192
             PER: precision:  37.69%; recall:  32.90%; FB1:  35.13  1608
Epoch [9/20] - Loss: 0.2623
validation error:
processed 51362 tokens with 5942 phrases; found: 5718 phrases; correct: 3226.
accuracy:  54.99%; (non-O)
accuracy:  91.36%; precision:  56.42%; recall:  54.29%; FB1:  55.33
             LOC: precision:  85.86%; recall:  62.17%; FB1:  72.12  1330
            MISC: precision:  71.46%; recall:  64.10%; FB1:  67.58  827
             ORG: precision:  49.14%; recall:  53.09%; FB1:  51.04  1449
             PER: precision:  36.98%; recall:  42.40%; FB1:  39.50  2112
Epoch [10/20] - Loss: 0.2519
validation error:
processed 51362 tokens with 5942 phrases; found: 5262 phrases; correct: 3086.
accuracy:  52.47%; (non-O)
accuracy:  91.59%; precision:  58.65%; recall:  51.94%; FB1:  55.09
             LOC: precision:  83.68%; recall:  65.05%; FB1:  73.20  1428
            MISC: precision:  72.64%; recall:  66.81%; FB1:  69.60  848
             ORG: precision:  54.62%; recall:  52.42%; FB1:  53.50  1287
             PER: precision:  33.67%; recall:  31.05%; FB1:  32.31  1699
Epoch [11/20] - Loss: 0.2380
validation error:
processed 51362 tokens with 5942 phrases; found: 5797 phrases; correct: 3229.
accuracy:  54.00%; (non-O)
accuracy:  91.25%; precision:  55.70%; recall:  54.34%; FB1:  55.01
             LOC: precision:  81.90%; recall:  66.03%; FB1:  73.12  1481
            MISC: precision:  70.90%; recall:  66.59%; FB1:  68.68  866
             ORG: precision:  50.79%; recall:  52.42%; FB1:  51.60  1384
             PER: precision:  33.83%; recall:  37.95%; FB1:  35.77  2066
Epoch [12/20] - Loss: 0.2313
validation error:
processed 51362 tokens with 5942 phrases; found: 5443 phrases; correct: 3330.
accuracy:  54.59%; (non-O)
accuracy:  91.61%; precision:  61.18%; recall:  56.04%; FB1:  58.50
             LOC: precision:  81.57%; recall:  68.43%; FB1:  74.42  1541
            MISC: precision:  72.21%; recall:  65.94%; FB1:  68.93  842
             ORG: precision:  54.06%; recall:  54.06%; FB1:  54.06  1341
```

```
              PER: precision:  43.05%; recall:  40.17%; FB1:  41.56  1719
Epoch [13/20] - Loss: 0.2247
validation error:
processed 51362 tokens with 5942 phrases; found: 5448 phrases; correct: 3308.
accuracy:  55.54%; (non-O)
accuracy:  91.77%; precision:  60.72%; recall:  55.67%; FB1:  58.09
              LOC: precision:  80.15%; recall:  68.37%; FB1:  73.80  1567
             MISC: precision:  75.03%; recall:  69.41%; FB1:  72.11  853
              ORG: precision:  55.00%; recall:  45.56%; FB1:  49.84  1111
              PER: precision:  41.78%; recall:  43.49%; FB1:  42.62  1917
Epoch [14/20] - Loss: 0.2153
validation error:
processed 51362 tokens with 5942 phrases; found: 5521 phrases; correct: 3226.
accuracy:  55.07%; (non-O)
accuracy:  91.79%; precision:  58.43%; recall:  54.29%; FB1:  56.29
              LOC: precision:  79.81%; recall:  68.86%; FB1:  73.93  1585
             MISC: precision:  70.17%; recall:  68.11%; FB1:  69.12  895
              ORG: precision:  54.26%; recall:  46.53%; FB1:  50.10  1150
              PER: precision:  37.49%; recall:  38.49%; FB1:  37.99  1891
Epoch [15/20] - Loss: 0.2109
validation error:
processed 51362 tokens with 5942 phrases; found: 5244 phrases; correct: 3283.
accuracy:  54.82%; (non-O)
accuracy:  91.86%; precision:  62.60%; recall:  55.25%; FB1:  58.70
              LOC: precision:  79.68%; recall:  68.32%; FB1:  73.56  1575
             MISC: precision:  71.86%; recall:  67.03%; FB1:  69.36  860
              ORG: precision:  55.08%; recall:  50.11%; FB1:  52.48  1220
              PER: precision:  46.44%; recall:  40.07%; FB1:  43.02  1589
Epoch [16/20] - Loss: 0.2058
validation error:
processed 51362 tokens with 5942 phrases; found: 5285 phrases; correct: 3317.
accuracy:  55.93%; (non-O)
accuracy:  92.03%; precision:  62.76%; recall:  55.82%; FB1:  59.09
              LOC: precision:  82.97%; recall:  68.43%; FB1:  75.00  1515
             MISC: precision:  72.47%; recall:  69.09%; FB1:  70.74  879
              ORG: precision:  57.44%; recall:  49.81%; FB1:  53.35  1163
              PER: precision:  43.69%; recall:  40.99%; FB1:  42.30  1728
Epoch [17/20] - Loss: 0.2001
validation error:
processed 51362 tokens with 5942 phrases; found: 5265 phrases; correct: 3308.
accuracy:  55.90%; (non-O)
accuracy:  91.97%; precision:  62.83%; recall:  55.67%; FB1:  59.03
              LOC: precision:  84.92%; recall:  65.92%; FB1:  74.23  1426
             MISC: precision:  68.81%; recall:  71.80%; FB1:  70.28  962
              ORG: precision:  55.74%; recall:  50.71%; FB1:  53.10  1220
              PER: precision:  45.56%; recall:  40.99%; FB1:  43.16  1657
Epoch [18/20] - Loss: 0.1967
validation error:
```

```
processed 51362 tokens with 5942 phrases; found: 5334 phrases; correct: 3472.
accuracy:  57.34%; (non-O)
accuracy:  92.18%; precision:  65.09%; recall:  58.43%; FB1:  61.58
              LOC: precision:  83.97%; recall:  67.88%; FB1:  75.08  1485
             MISC: precision:  75.44%; recall:  69.96%; FB1:  72.59  855
              ORG: precision:  56.91%; recall:  56.23%; FB1:  56.56  1325
              PER: precision:  49.49%; recall:  44.84%; FB1:  47.05  1669
Epoch [19/20] - Loss: 0.1907
validation error:
processed 51362 tokens with 5942 phrases; found: 5652 phrases; correct: 3506.
accuracy:  57.48%; (non-O)
accuracy:  91.84%; precision:  62.03%; recall:  59.00%; FB1:  60.48
              LOC: precision:  79.36%; recall:  71.15%; FB1:  75.03  1647
             MISC: precision:  78.36%; recall:  70.28%; FB1:  74.10  827
              ORG: precision:  57.29%; recall:  50.41%; FB1:  53.63  1180
              PER: precision:  43.79%; recall:  47.50%; FB1:  45.57  1998
Epoch [20/20] - Loss: 0.1868
validation error:
processed 51362 tokens with 5942 phrases; found: 5710 phrases; correct: 3461.
accuracy:  58.29%; (non-O)
accuracy:  92.03%; precision:  60.61%; recall:  58.25%; FB1:  59.41
              LOC: precision:  83.07%; recall:  70.50%; FB1:  76.27  1559
             MISC: precision:  75.51%; recall:  68.87%; FB1:  72.04  841
              ORG: precision:  57.02%; recall:  53.02%; FB1:  54.95  1247
              PER: precision:  39.75%; recall:  44.52%; FB1:  42.00  2063
```

[1191]:
```python
# 3 more epochs
for epoch in range(3):
    model.train()
    total_loss = 0.0

    # Iterate over your training data in batches
    for batch_idx, (inputs, targets) in enumerate(train_loader):
        # Zero the gradients
        optimizer.zero_grad()

        # Forward pass + src_padding_mask
        src_padding_mask = (inputs == 0).float()
        outputs = model(inputs, src_padding_mask= src_padding_mask)

        # Flatten the outputs and targets for the loss calculation
        outputs = outputs.view(-1, 9)
        targets = targets.view(-1)

        # Calculate the loss
        loss = criterion(outputs, targets)
```

```
        # Backpropagation
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    # Print the average loss for this epoch
    avg_loss = total_loss / len(train_loader)
    print(f"Epoch [{epoch+22}/{25}] - Loss: {avg_loss:.4f}")

    print('validation error: ')
    precision, recall, f1 = eval(model, val_loader)
```

```
Epoch [22/25] - Loss: 0.1733
validation error:
processed 51362 tokens with 5942 phrases; found: 5506 phrases; correct: 3568.
accuracy:  59.12%; (non-O)
accuracy:  92.24%; precision:  64.80%; recall:  60.05%; FB1:  62.33
              LOC: precision:  83.40%; recall:  69.73%; FB1:  75.96  1536
             MISC: precision:  76.79%; recall:  71.04%; FB1:  73.80  853
              ORG: precision:  57.69%; recall:  53.69%; FB1:  55.62  1248
              PER: precision:  48.80%; recall:  49.51%; FB1:  49.15  1869
Epoch [23/25] - Loss: 0.1700
validation error:
processed 51362 tokens with 5942 phrases; found: 5337 phrases; correct: 3540.
accuracy:  56.79%; (non-O)
accuracy:  92.03%; precision:  66.33%; recall:  59.58%; FB1:  62.77
              LOC: precision:  84.71%; recall:  69.95%; FB1:  76.62  1517
             MISC: precision:  76.77%; recall:  70.61%; FB1:  73.56  848
              ORG: precision:  55.55%; recall:  57.87%; FB1:  56.68  1397
              PER: precision:  52.57%; recall:  44.95%; FB1:  48.46  1575
Epoch [24/25] - Loss: 0.1684
validation error:
processed 51362 tokens with 5942 phrases; found: 5823 phrases; correct: 3586.
accuracy:  58.87%; (non-O)
accuracy:  91.86%; precision:  61.58%; recall:  60.35%; FB1:  60.96
              LOC: precision:  84.38%; recall:  69.41%; FB1:  76.16  1511
             MISC: precision:  76.08%; recall:  68.66%; FB1:  72.18  832
              ORG: precision:  54.89%; recall:  56.08%; FB1:  55.48  1370
              PER: precision:  43.89%; recall:  50.27%; FB1:  46.86  2110
```

[1192]:
```
print(f"Validation: precision = {precision}, recall = {recall}, f1 = {f1}")
```

```
Validation: precision = 61.58337626652928, recall = 60.35005048805117, f1 =
60.96047598810029
```

[1193]:
```
# SAVE THE MODEL
torch.save(model.state_dict(), 'task3.pth')
```

[1186]:
```
#evaluation
def eval(model, loader):
    model.eval()
    all_preds, all_labels = [], []
    with torch.no_grad():
        for batch in loader:
            inputs, targets = batch
            #get rid of paddings on targets
            label_unpad = targets
            mask = label_unpad != 9
            label_unpad = label_unpad[mask]
            src_padding_mask = (inputs == 0).float()
                # print('size match:', src_padding_mask.size() == inputs.size())
            outputs = model(inputs,src_padding_mask=src_padding_mask)
            _, preds = torch.max(outputs, -1)
            #get rid of paddings on pred
            preds = preds[mask]

            preds_converted = [reversed_ner_tags[elem.item()] for elem in preds]
            targets_converted = [reversed_ner_tags[elem.item()] for elem in␣
 ↪label_unpad]
            all_preds.extend(preds_converted)
            all_labels.extend(targets_converted)
    # all_preds = list(chain.from_iterable(all_preds))
    # all_labels = list(chain.from_iterable(all_labels))
    # all_labels = torch.cat(all_labels)
    # all_preds = itertools.chain(*all_preds)
    # all_labels =itertools.chain(*all_labels)
    result = evaluate(all_labels, all_preds,verbose=True)
    precision, recall, f1 = result[0], result[1],result[2]
    return precision, recall, f1
```

[1194]:
```
print('Test: ')
precision, recall, f1 = eval(model, test_loader)
```

```
Test:
processed 46435 tokens with 5648 phrases; found: 5360 phrases; correct: 2830.
accuracy:  50.48%; (non-O)
accuracy:  89.56%; precision:  52.80%; recall:  50.11%; FB1:  51.42
              LOC: precision:  79.85%; recall:  64.15%; FB1:  71.14  1340
             MISC: precision:  67.76%; recall:  61.97%; FB1:  64.73  642
              ORG: precision:  49.84%; recall:  45.88%; FB1:  47.77  1529
              PER: precision:  30.45%; recall:  34.82%; FB1:  32.49  1849
```

[1195]:
```
print(f"Test: precision = {precision}, recall = {recall}, f1 = {f1}")
```

```
Test: precision = 52.79850746268657, recall = 50.106232294617556, f1 =
51.417151162790695
```

### 0.2.1 Solution to task 3

1. Hyperparameters:

- embedding_dim = 100
- hidden_dim = 256
- output_dim = 128
- num_layers = 1
- dropout = 0.33
- optimizer learning rate= 0.001
- ignore_index = 9
- batch_size = 32

2. Solution: Same as task, we still use input_ids as the input. The dataloader will pad 0 to input and 9 to ner_tags. Next, the first layer of the model is an embedding layer, which convert each token into 128-d vector. The positional encoder is a self-attention layer which will generate a sequence as output. For src_padding_mask, it will identify all the padded values and get rid of their impact. Next, the bacthes will be thrown to the transformer encoder and a FFN to predict the results.
3. Questions and answers:

- What is the precision, recall, and F1 score on the validation data?
- precision = 61.58337626652928, recall = 60.35005048805117, f1 = 60.96047598810029
- What are the precision, recall, and F1 score on the test data?
- precision = 52.79850746268657, recall = 50.106232294617556, f1 = 51.417151162790695- What is the reason behind the poor performance of the transformer?
- At first, the transformer typically require big amout of data. Since the word2idx is too small, it cannot generalize well onto the new data. Secondly, the other problem of the small dataset is that the model will probably overfit.